개발문서

작성자 : 한창신

Ellie 개발문서

유니티 팀 프로젝트

Monster & Behaviour Tree

게임 영상 보러가기 [유튜브]

목차

1. 기획내용

1-1. 기획팀 기획 내용

- 몬스터 기획서

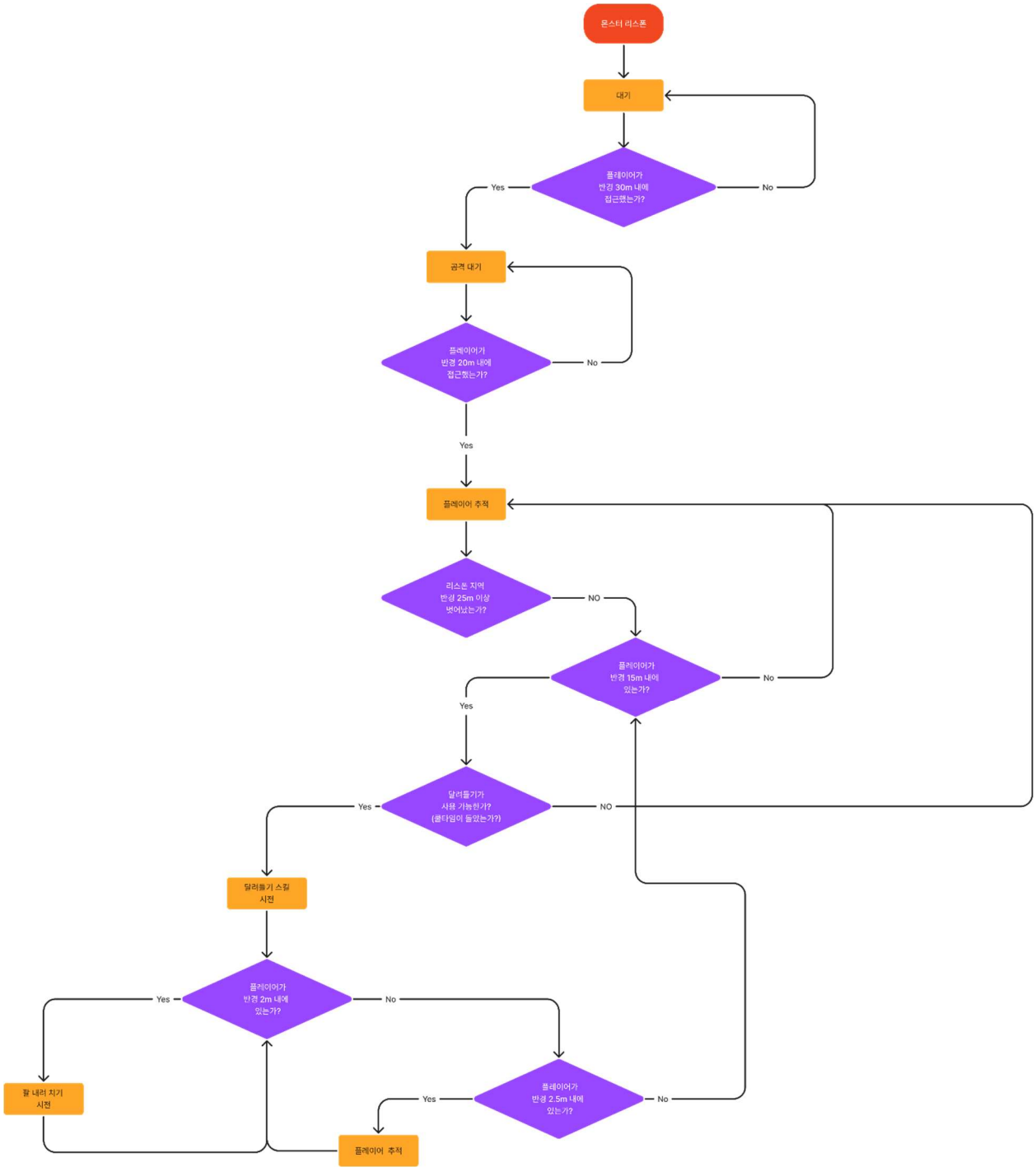
평범한 스켈레톤		
대분류	소분류	설명
정보	체력	5
	공격력	1
	공격 범위	中
	공격속도	2
	기본 이동속도	2m/s
속성	등급	일반
	타입	지상
	계열	해골 계열
	속성	무 속성
	공격 방식	근거리
	공격 타입	선공
	등장장소	미정 (추후 맵 좌표 값으로 설정할 것)
	서식지	던전 내 아무 곳
	기획특성	던전 곳곳에서 흔히 등장하는 몬스터
	습성	평범하고 느리게 움직이는 해골 이미지
	배경 스토리	마법사의 무덤 근처에서 죽은 자들의 영혼은 마석의 마력에 영혼이 묶여 시체에 남게 된다. 그들은 처음엔 불멸을 얻은 줄 알고 좋아했으나 육체는 점점 썩어 없어지고 잠들지도 먹지도 못한 채 시간을 보냈다. 시간이 지나 그들은 이성을 잃고 미쳐버리며 마법사의 무덤 근처를 돌아다니는 몬스터로 남게 되었다.
	행동	주변에 인기척이 들리면 움직이기 시작한다. 관절 하나, 하나 삐걱거리며 움직인다. 움직임이 멈추면 갑자기 달려든다.
패턴		설명
공격	1:팔 내려치기	팔로 적을 내려쳐 플레이어에게 1의 피해를 입히고 약경직 상태로 만든다.
이동	1: 달려들기	플레이어를 향해 4m/s로 달려간다.
애니메이션		설명
기본대기		누워있는 상태
공격대기	1	누워있다가 플레이어가 근접 및 공격 시 일어나서 정지 상태
	2	근접 시 서서히 맵을 돌아다닌다
이동	1	천천히 플레이어를 향해 걸어 온다.
	2	갑자기 팔을 들어 올리며 플레이어를 향해 달려든다.
공격	1	두 팔을 들어 올렸다 몸과 함께 내려치며 공격한다.
맞는 동작	1	고개가 뒤로 젖혀지며 팔을 들어 올라가고 몸 전체가 뒤로 약간 밀리는 형태
사망	1	뼈들이 우수수 떨어지는 부서지는 형태

모험가 스켈레톤		
대분류	소분류	설명
정보	체력	5
	공격력	1
	공격범위	中
	공격속도	2
	기본 이동속도	2m/s
속성	등급	일반
	타입	지상
	계열	해골 계열
	속성	무속성
	공격 방식	근거리
	공격 타입	선공
	등장장소	미정 (추후 맵 좌표 값으로 설정할 것)
	서식지	탐사구역 (필드 외곽)
	기획특성	탐사구역에서 등장하는 근거리 해골계열 몬스터
	습성	평범하고 느리게 움직이는 해골에 장비가 입혀진 이미지 평범한 스켈레톤 보다 단단하고 위협적인 공격을 한다.
	배경 스토리	마법사의 무덤 근처에서 죽은 모험가들의 영혼은 마석의 마력에 영혼이 묶여 시체에 남게 된다. 죽은 줄도 모르고 던전을 탐험하던 그들의 육체는 점점 썩어 없어지고 잠들지도 먹지도 못한 채 시간을 보냈다. 시간이 지나 그들은 이성을 잃고 미쳐버리며 마법사의 무덤 근처를 돌아다니는 몬스터로 남게 되었다.
	행동	주변에 인기척이 들리면 움직이기 시작한다. 관절 하나, 하나 삐걱거리며 움직인다. 움직임이 멈추면 갑자기 달려든다.
패턴		설명
공격	1:팔 내려치기	팔로 적을 내려쳐 플레이어에게 1의 피해를 입히고 약경직 상태로 만든다.
	2: 휘두르기	오른 손에 들고있는 무기를 가로로 휘두르며 플레이어에게 1의 피해를 입히고 약경직 상태로 만든다.
이동	1: 달려들기	플레이어를 향해 4m/s로 달려간다.
애니메이션		설명
기본대기		누워있는 상태
공격대기	1	누워있다가 플레이어가 근접 및 공격 시 일어나서 정지 상태
	2	근접 시 서서히 맵을 돌아다닌다
이동	1	천천히 플레이어를 향해 걸어 온다. (기본 이동)
	2	갑자기 팔을 들어 올리며 플레이어를 향해 달려든다.
공격	1	두 팔을 들어 올렸다 내려쳐 공격한다.
	2	오른 손에 들고 있는 무기를 앞을 향해 휘두르며 공격한다.
맞는 동작	1	고개가 뒤로 젖혀지며 팔을 들어 올라가고 몸 전체가 뒤로 약간 밀리는 형태
사망	1	뼈들이 우수수 떨어지는 부서지는 형태

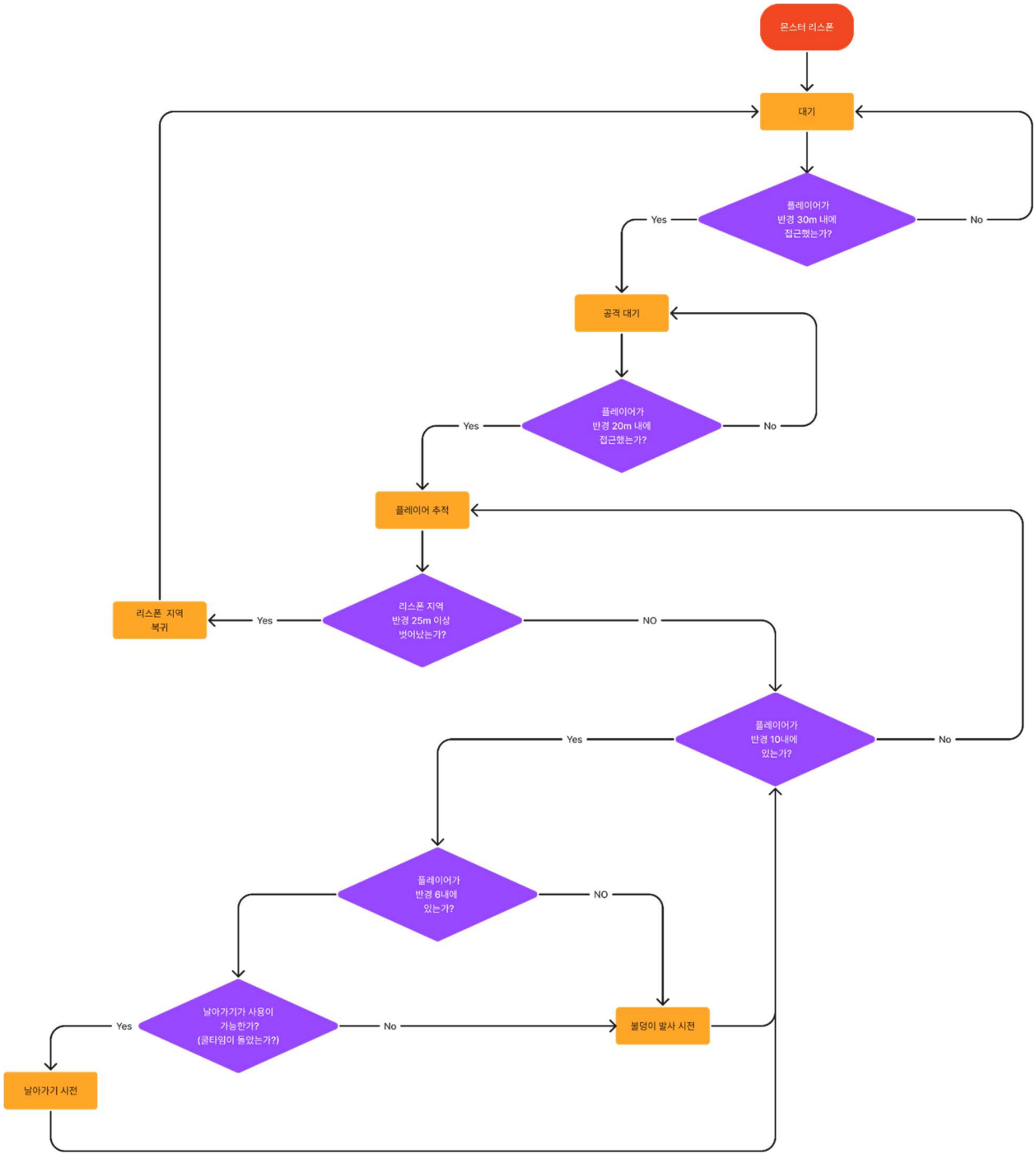
마법사 스켈레톤		
대분류	소분류	설명
정보	체력	3
	공격력	1
	공격범위	中
	공격속도	3
	기본 이동속도	3m/s
속성	등급	일반
	타입	공중
	계열	해골 계열
	속성	무속성
	공격 방식	마법
	공격 타입	선공
	등장장소	미정 (추후 맵 좌표 값으로 설정할 것)
	서식지	탐사구역 (필드 외곽)
	기획특성	탐사구역에서 등장하는 마법공격 해골계열 몬스터
	습성	평범하고 느리게 움직이는 해골에 마법사 장비가 입혀진 이미지 다른 스켈레톤과 달리 마법 공격을 사용한다. 마법으로 몸을 띄워 다른 해골들에 비해 움직임이 빠르다.
	배경 스토리	오래전 흑마법사의 성에서 '흑마법사 제레프'와 전투를 하다 목숨을 잃은 마법사들이 마석의 마력에 영혼이 묶여 시체에 남게 된다. 그들은 죽어서도 제레프와 맞서 싸웠지만 끝내 패배하고 다음을 기약하며 연대드로 살게된다. 하지만 그들의 육체는 점점 썩어 없어지고 잠들지도 먹지도 못한 채 시간을 보냈다. 시간이 지나 그들은 이성을 잃고 미쳐버리며 마법사의 무덤 근처를 돌아다니는 몬스터로 남게 되었다.
	행동	주변에 인기척이 들리면 움직이기 시작한다. 몸이 붕 뜨며 하늘을 떠다닌다. 움직임이 멈추면 갑자기 마법을 사용한다.
패턴		설명
공격	1:불덩이 발사	2초 간 마력을 모아 플레이어를 향해 1에 피해를 주는 불덩이를 10m 만큼 발사한다.
이동	1: 날아가기	플레이어가 일정 범위 다가올 경우 4m 뒤로 날아간다.
애니메이션		설명
기본대기		누워있는 상태
공격대기	1	누워있다가 플레이어가 근접 및 공격 시 공중에서 뜬 상태에서 정지
	2	근접 시 서서히 맵을 돌아다닌다
이동	1	공중에 떠다니며 플레이어 근처에서 거리를 둔다.
	2	플레이어가 일정 범위 다가올 경우 거리를 벌리며 날아간다.
마법시전	1	두 팔을 내밀며 불덩이를 발사한다.
맞는 동작	1	고개가 뒤로 젖혀지며 팔을 들어 올라가고 몸 전체가 뒤로 약간 밀리는 형태
사망	1	공중에서 뼈들이 우수수 떨어지는 부서지는 형태

공대원 스켈레톤		
대분류	소분류	설명
정보	체력	7
	공격력	2
	공격범위	中
	공격속도	3
	기본 이동속도	3m/s
관련 돌멩이	약점 돌멩이	목발 돌멩이, 불 속성 돌멩이, 날카로운 돌멩이
	무시 돌멩이	
속성	등급	일반
	타입	지상
	계열	해골 계열
	속성	무속성
	공격 방식	근거리
	공격 타입	선공
	등장장소	미정 (추후 맵 좌표 값으로 설정할 것)
	서식지	탐사구역 (필드 외곽)
	기획특성	탐사구역에서 등장하는 근거리 해골계열 몬스터
	습성	모험가 스켈레톤보다 강력한 모험가 스켈레톤 모험가 스켈레톤 보다 단단하고 강한 공격을 한다.
	배경 스토리	마법사의 무덤 깊은 곳에서 죽은 모험가들의 영혼은 마석의 마력에 영혼이 묶여 시체에 남게 된다. 죽은 줄도 모르고 던전을 탐험하던 그들의 육체는 점점 썩어 없어지고 잠들지도 먹지도 못한 채 시간을 보냈다. 시간이 지나 그들은 이성을 잃고 미쳐버리며 마법사의 무덤 근처를 돌아다니는 몬스터로 남게 되었다.
	행동	주변에 인기척이 들리면 움직이기 시작한다. 관절 하나, 하나 삐걱거리며 움직인다. 움직임이 멈추면 갑자기 달려든다.
패턴		설명
공격	1:팔 내려치기	팔로 적을 내려쳐 플레이어에게 3의 피해를 입히고 약경직 상태로 만든다.
이동	1: 달려들기	플레이어를 향해 5m/s로 달려간다.
애니메이션		설명
기본대기		누워있는 상태
공격대기	1	누워있다가 플레이어가 근접 및 공격 시 일어나서 정지 상태
	2	근접 시 서서히 맵을 돌아다닌다
이동	1	천천히 플레이어를 향해 걸어 온다. (기본 이동)
	2	갑자기 팔을 들어 올리며 플레이어를 향해 달려든다.
공격	1	두 팔을 들어 올렸다 내려쳐 공격한다.
맞는 동작	1	고개가 뒤로 젖혀지며 팔을 들어 올라가고 몸 전체가 뒤로 약간 밀리는 형태
사망	1	뼈들이 우수수 떨어지는 부서지는 형태

- 근접공격 다이어그램


몬스터 리스폰
대기
플레이어가
반경 30m 내에
접근했는가?
Yes
No
공격 대기
플레이어가
반경 20m 내에
접근했는가?
No
Yes
플레이어 추적
리스폰 지역
반경 25m 이상
벗어났는가?
NO
플레이어가
반경 15m 내에
있는가?
No
Yes
달려들기가
사용 가능한가?
(쿨타임이 돌았는가?)
Yes
NO
달려들기 스킬
시전
플레이어가
반경 2m 내에
있는가?
Yes
No
팔 내려 치기
시전
플레이어가
반경 2.5m 내에
있는가?
Yes
No
플레이어 추적

- 원거리 몬스터 다이어그램



1-2. 개발팀 기획 요청 내용

1-2.1 몬스터 공통 Stat

몬스터 공통 스텟 내용						
구분	값				자료형	기타
	평범한 스켈레톤	모험가 스켈레톤	마법사 스켈레톤	공대원 스켈레톤		
체력					float	최대(시작) 체력
속도					float	일반 걷기 속도 (초당 미터)
회전 속도					float	몬스터 초당 n도 회전
플레이어 감지					float	미터 단위
플레이어 추격 감지					float	
초과이동 거리					float	
스폰 지역 복귀 속도					float	초당 미터
플레이어 추적 멈춤 거리					float	몬스터가 추적 완료되는 거리
약점 비율					float	약점 맞을 시 데미지의 몇배 받는지
리젠 시간					float	죽은 후 리젠 타임
드랍 아이템					list<int>	아이템 드랍 번호 입력

1-2.2 몬스터 공격

- 공격 여부 표시

스킬		몬스터			
기획팀	개발팀	평범한 스켈레톤	모험가 스켈레톤	마법사 스켈레톤	공대원 스켈레톤
달려들기	RunToPlayer	O	O		O
팔내려치기	MeleeAttack	O	O		O
파이어볼	Fireball			O	
달아나기	Flee			O	
무기 공격	WeaponAttack		O		O

- 근접공격

몬스터 근접 공격 내용						
구분	값				자료형	기타
	평범한 스켈레톤	모험가 스켈레톤	마법사 스켈레톤	공대원 스켈레톤		
데미지					int	
쿨타임					float	
공격 가능 거리					float	몬스터가 플레이어와의 거리가 얼만큼 되어야 공격을 하는지
공격 지속 시간					float	공격이 얼만큼 지속되는지
애니메이션 대기 시간					float	공격 애니메이션을 실행 후 공격하는 타이밍
공격 위치					Vector3	몬스터 기준 공격이 되는 위치(몬스터 기준 x, y, z 미터 단위)
공격 범위					Vector3	공격위치 범위 x, y, z (미터단위)
* x : 몬스터 오른쪽, y : 몬스터 위쪽, z : 몬스터전방						
* 공격 위치와 범위 유니티에서 보이게 해드리니 그거 보시고 작성하시면 편하실거에요!						

- 달려들기

몬스터 달려들기						
구분	값				자료형	기타
	평범한 스켈레톤	모험가 스켈레톤	마법사 스켈레톤	공대원 스켈레톤		
쿨타임					int	
공격 가능 거리					float	
공격 지속 시간					float	
속도					float	달려들기 속도
멈춤 거리					float	
속도는 기존 속도의 배율로 해도 되고 속도 값을 넣어주셔도 됩니다. 표시만 부탁드려요						

1-2.3 몬스터 행동 우선순위

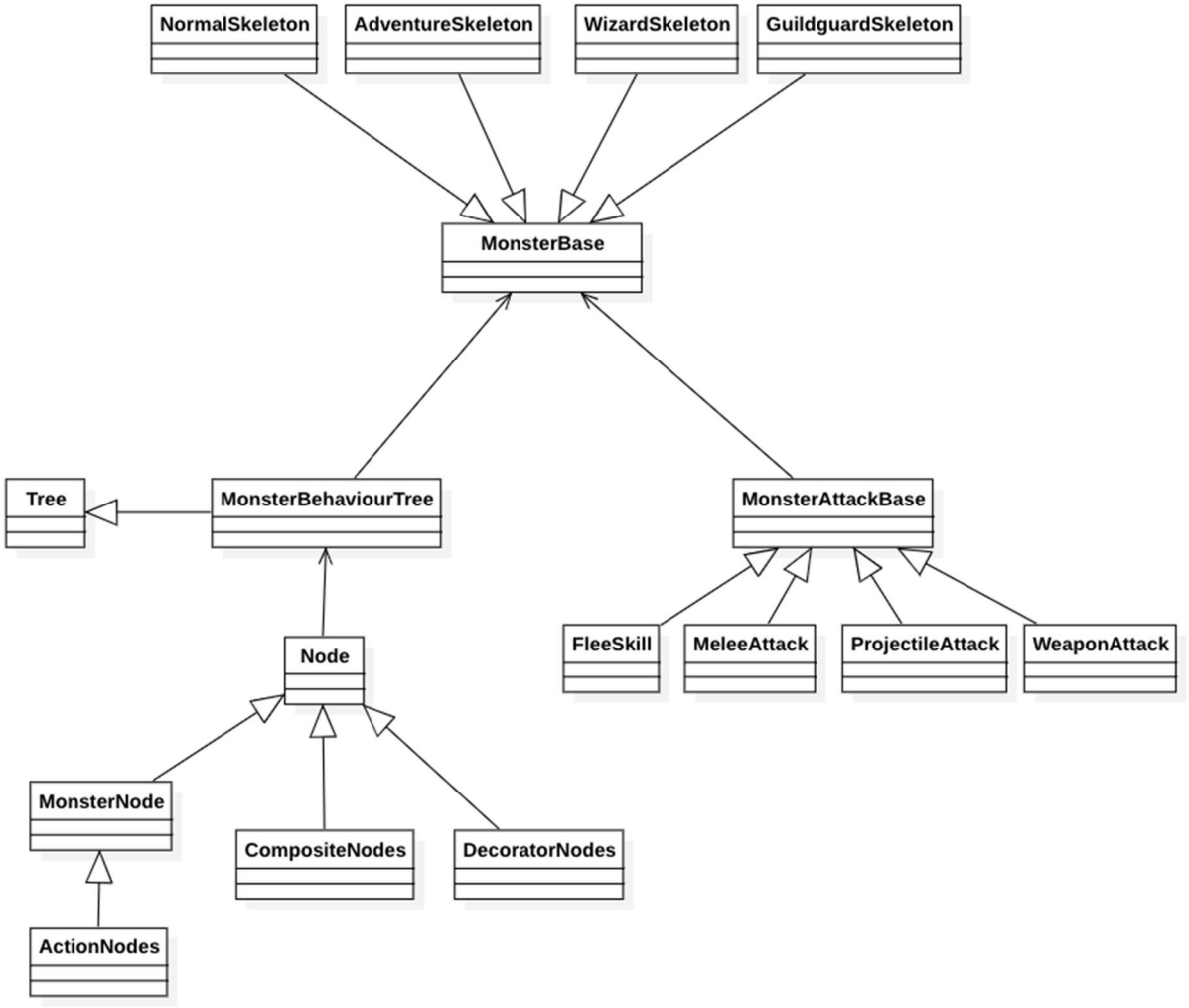
	구분	평범한 스켈레톤	모험가 스켈레톤	마법사 스켈레톤	공대원 스켈레톤
우선순위	플레이어 추적	4	5	4	5
	초과이동 복귀	1	1	1	1
	달려들기	3	4		4
	팔내려치기	2	3		3
	파이어볼			2	
	달아나기			3	
	무기공격		2		2

* 몬스터가 하는 행동에 우선순위가 필요합니다. 예를 들어 몬스터가 플레이어에게 달려들기와 팔 내려치기를 동시에 할 수 있을 때 어느 행동을 먼저 할지 정해주시면 그에 맞게 개발 할 수 있습니다.
* 제가 개발하면서 우선순위 임의로 정했고, 확인해보시고 수정내용 있으시면 말씀 부탁드립니다.

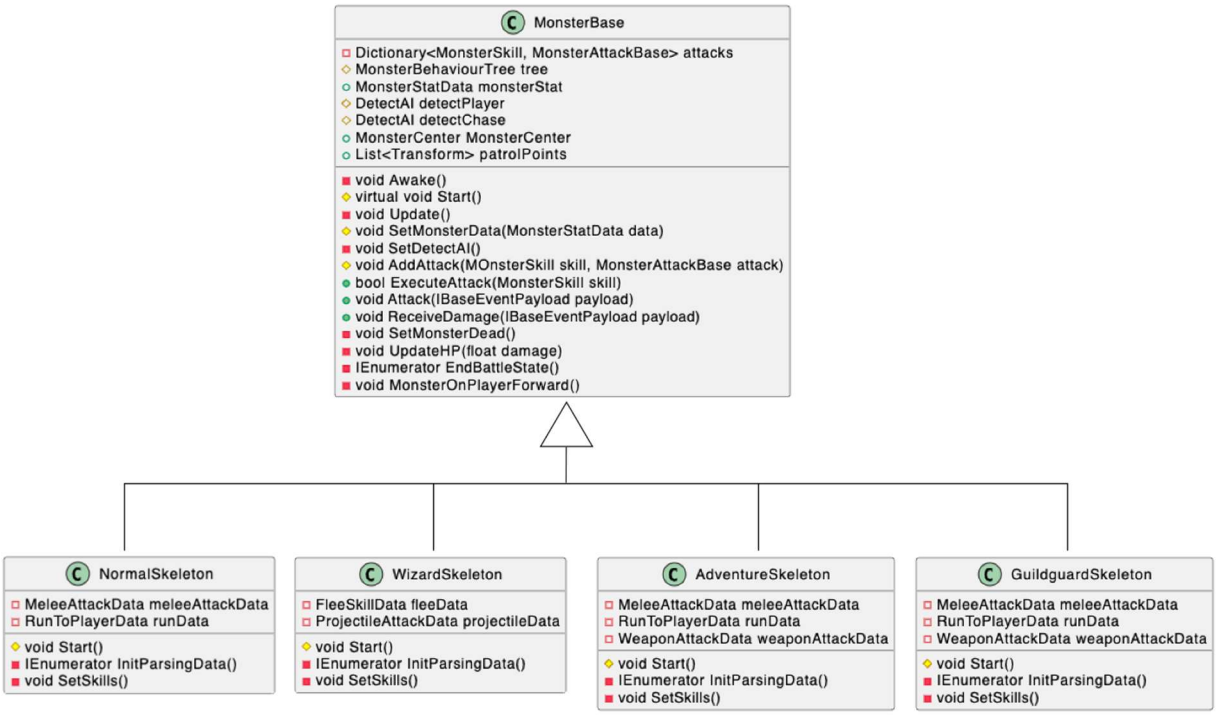
2. 몬스터 설계

전체 몬스터 설계 코드 확인하기

2-1. Diagram



2-2. Monster



2-2.1 MonsterBase

- 공통 사항

모든 몬스터의 부모 클래스로 몬스터가 가지고 있어야 하는 공격, Behaviour Tree, 몬스터 Stat, 플레이어를 인지하는 AI 와 쫓아가는 AI, Patrol Points 를 기본적으로 가지고 있다.

- 공격

모든 공격의 부모 클래스 MonsterAttackBase 를 사용하여 몬스터의 공격을 AddAttack()함수를 통해 추가할 수 있고, ExecueteAttack 을 통해 공격을 실행할 수 있다.

AddAttack 으로 공격을 추가하면 몬스터 자식으로 해당 공격의 script 를 가지고 있는 empty object 가 몬스터 자식으로 추가된다.

```
namespace Scripts.BehaviourTrees.Monster
{
    public class MonsterBase : MonoBehaviour, ICombatant
    {
        private Dictionary<MonsterSkill, MonsterAttackBase> attacks;
        [SerializeField] protected MonsterBehaviourTree tree;
        public MonsterStatData monsterStat;
        protected DetectAI detectPlayer;
        protected DetectAI detectChase;
```

```
[SerializeField] public MonsterCenter MonsterCenter { get; private set; }
[SerializeField] public List<Transform> patrolPoints;

private void Awake() { }
protected virtual void Start()
{
    tree.AddMonsterData<GameObject>(MonsterData.v3SpawnPosition, transform.position);
}
private void Update() { }

// >> : Set Datas
protected void SetMonsterData(MonsterStatData data) { }
private void SetDetectAI()
{
    GameObject detectPlayerObj = new GameObject("DetectPlayer");
    detectPlayerObj.transform.SetParent(transform);
    detectPlayerObj.transform.localPosition = Vector3.zero;
    detectPlayerObj.transform.localRotation = Quaternion.Euler(Vector3.zero);
    detectPlayerObj.transform.localScale = Vector3.one;
    detectPlayer = detectPlayerObj.AddComponent<DetectAI>();

    GameObject detectChaseObj = new GameObject("DetectChase");
    detectChaseObj.transform.SetParent(transform);
    detectChaseObj.transform.localPosition = Vector3.zero;
    detectChaseObj.transform.localRotation = Quaternion.Euler(Vector3.zero);
    detectChaseObj.transform.localScale = Vector3.one;
    detectPlayer = detectChaseObj.AddComponent<DetectAI>();
    detectChaseObj.AddComponent<SphereCollider>();
}
protected void AddAttack(MonsterSkill skill, MonsterAttackBase attack)
{
    GameObject attackObj = new GameObject(skill.ToString());
    attackObj.transform.SetParent(transform);
    attackObj.transform.localPosition = Vector3.zero;
    attackObj.transform.localRotation = Quaternion.Euler(Vector3.zero);
    attackObj.transform.localScale = Vector3.one;

    switch (skill)
    {
        case MonsterSkill.Melee:
            attack = attackObj.AddComponent<MonsterMeleeAttack>();
            break;
        case MonsterSkill.Projectile:
            attack = attackObj.AddComponent<MonsterProjectileAttack>();
            break;
        case MonsterSkill.Weapon:
            attack = attackObj.AddComponent<MonsterWeaponeAttack>();
            break;
        case MonsterSkill.Flee:
```

```
                    attack = attackObj.AddComponent<MonsterFleeSkill>();
                    break;
                }
                if (attack == null)
                {
                    Debug.LogFormat("{0} Failed To Add Skill {1}", transform.name, skill.ToString());
                    return;
                }

                attacks.Add(skill, attack);
            }

            // >> : Battles
            public bool ExecuteAttack(MonsterSkill skill)
            {
                MonsterAttackBase attack;
                if (attacks.TryGetValue(skill, out attack))
                {
                    attack.ExecuteAttack();
                    return true;
                }

                Debug.LogFormat("Trying To Access Attack Does Not Have : {0}, {1}", transform.name, skill.ToString());
                return false;
            }
            public void Attack(IBaseEventPayload payload) { }

            public void ReceiveDamage(IBaseEventPayload payload) { }

            // >> : MonsterDamaged or Dead
            private void SetMonsterDead() { }

             private void UpdateHP(float damage) { }

            // >> : Billboard
            private IEnumerator EndBattleState() { }
            private void MonsterOnPlayerForward() { }
             private void HideBillboard() { }
             private void ShowBillboard() { }
        }
}
```

* 전체 코드 확인하기

2-2.2 Monsters

몬스터 정보를 파싱을 위한 Parsing Number 와 해당 몬스터가 가지고 있는 공격 정보를 가지고 있다.

Parsing 이후 MonsterBase 에 있는 attack Dictionary 에 해당 몬스터가 가지고 있는 공격을 추가 해준다.

- 평범한 스켈레톤 (NormalSkeleton)

```
using System.Collections;
using Assets.Scripts.Managers;

namespace Scripts.BehaviourTrees.Monster
{
    public class NormalSkeleton : MonsterBase
    {
        private enum ParsingData
        {
            MonsterStat = 1900,
            MeleeAttack = 2000,
            RunToPlayer = 2005,
        }

        private MeleeAttackData meleeAttackData;
        private RunToPlayerData runData;

        protected override void Start()
        {
            base.Start();
            StartCoroutine(InitParsingData());
        }
        private IEnumerator InitParsingData()
        {
            yield return DataManager.Instance.CheckIsParseDone();

            monsterStat = DataManager.Instance.GetIndexData<MonsterStatData,
                          MonsterStatDataParsingInfo>((int)ParsingData.MonsterStat);
            meleeAttackData = DataManager.Instance.GetIndexData<MeleeAttackData,
                          MonsterAttackDataparsingInfo>((int)ParsingData.MeleeAttack);
            runData = DataManager.Instance.GetIndexData<RunToPlayerData,
                          MonsterAttackDataparsingInfo>((int)ParsingData.RunToPlayer);

            SetSkills();
            SetMonsterData(monsterStat);

            tree.AddMonsterData<MonsterStatData>(MonsterData.MonsterStat, monsterStat);
            tree.AddMonsterData<MeleeAttackData>(MonsterData.Melee, meleeAttackData);
```

```
            tree.AddMonsterData<RunToPlayerData>(MonsterData.RunToPlayer, runData);

        }
        private void SetSkills()
        {
            MonsterMeleeAttack meleeAttack = new();
            meleeAttack.SetInitialData(meleeAttackData);
            AddAttack(MonsterSkill.Melee, meleeAttack);
        }
    }
}
```

- 마법사 스켈레톤 (WizardSkeleton)

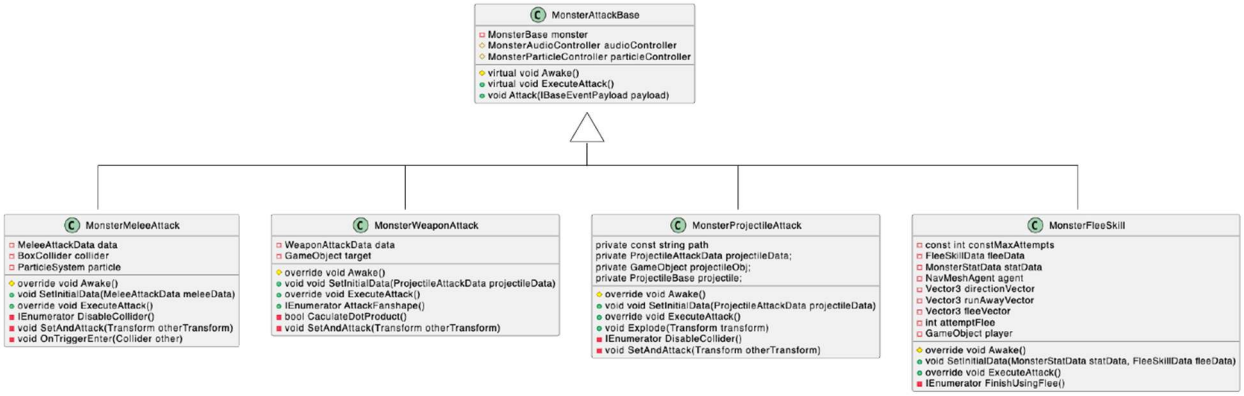
```
using System.Collections;
using Assets.Scripts.Managers;

namespace Scripts.BehaviourTrees.Monster
{
    public class WizardSkeleton : MonsterBase
    {
        private enum ParsingData
        {
            MonsterStat = 1902,
            Flee = 2008,
            ProjectileAttack = 2009,
        }

        private FleeSkillData fleeData;
        private ProjectileAttackData projectileData;

        protected override void Start()
        {
            base.Start();
            StartCoroutine(InitParsingData());
        }
        private IEnumerator InitParsingData()
        {
            yield return DataManager.Instance.CheckIsParseDone();

            monsterStat = DataManager.Instance.GetIndexData<MonsterStatData,
                          MonsterStatDataParsingInfo>((int)ParsingData.MonsterStat);
            fleeData = DataManager.Instance.GetIndexData<FleeSkillData,
                       MonsterAttackDataparsingInfo>((int)ParsingData.Flee);
            projectileData = DataManager.Instance.GetIndexData<ProjectileAttackData,
                             MonsterAttackDataparsingInfo>((int)ParsingData.ProjectileAttack);

            SetSkills();
            SetMonsterData(monsterStat);
```

```
            tree.AddMonsterData<MonsterStatData>(MonsterData.MonsterStat, monsterStat);
            tree.AddMonsterData<FleeSkillData>(MonsterData.Melee, fleeData);
            tree.AddMonsterData<ProjectileAttackData>(MonsterData.RunToPlayer, projectileData);
        }

        private void SetSkills()
        {
            MonsterFleeSkill fleeAttack = new();
            fleeAttack.SetInitialData(monsterStat, fleeData);
            AddAttack(MonsterSkill.Flee, fleeAttack);

            MonsterProjectileAttack projectileAttack = new();
            projectileAttack.SetInitialData(projectileData);
            AddAttack(MonsterSkill.Projectile, projectileAttack);
        }
    }
}
```

* 다른 몬스터 script 확인하기

3. 몬스터 공격

3-1. Diagram



3-2. Monster Attacks

3-2.1 MonsterAttackBase

모든 공격의 부모 클래스로 공격을 실행하는 ExecuteAttack 을 가상함수로 가지고 있어 자식 클래스에서 override 하여 사용한다

```
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class MonsterAttackBase : MonoBehaviour
    {
        private MonsterBase monster;
        protected MonsterAudioController audioController;
        protected MonsterParticleController particleController;

        protected virtual void Awake()
        {
            if (audioController == null)
                audioController = GetComponent<MonsterAudioController>();
            if (particleController == null)
                particleController = GetComponent<MonsterParticleController>();
        }

        public virtual void ExecuteAttack() {}

        public void Attack(IBaseEventPayload payload)
        {
            monster.Attack(payload);
        }
    }
}
```

3-2.2 Monster Skills

- MonsterMeleeAttack (BoxcolliderAttack)

설정된 크기와 위치에 BoxCollider 을 ExecuteAttack 을 통해 해당 Collider 을 Enable, Disable 하는 방식으로 공격

각 Monster 에서 설정된 MeleeAttack 정보를 토대로 BoxCollider 의 위치, 크기, 지속시간 등 필요한 정보를 모두 설정한다

```
using System.Collections;
using Assets.Scripts.Combat;
using Assets.Scripts.StatusEffects;
using Channels.Combat;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class MonsterMeleeAttack : MonsterAttackBase
    {
        private MeleeAttackData data;
        private BoxCollider collider;
        private ParticleSystem particle;

        protected override void Awake()
        {
            base.Awake();

            // Collider Setting
            if (collider == null)
                collider = gameObject.AddComponent<BoxCollider>();
            collider.isTrigger = true;
            collider.size = data.colliderSize;
            gameObject.transform.localPosition = data.colliderOffset;
            collider.enabled = false;
        }
        public void SetInitialData(MeleeAttackData meleeData)
        {
            data = meleeData;
        }
        public override void ExecuteAttack()
        {
            if(data==null)
                Debug.LogFormat("{0} Has Not Initialized MeleeAttack : {0}, MeleeAttack" + transform.name);
            collider.enabled = true;
            StartCoroutine(DisableCollider());
        }
        private IEnumerator DisableCollider()
```

```
        {
            yield return new WaitForSeconds(data.colliderDuration);
            collider.enabled = false;
        }
        private void OnTriggerEnter(Collider other)
        {
            if(other.CompareTag("Player"))
            {
                if(other.gameObject.GetComponent<ICombatant>()!=null)
                {
                    audioController.PlayAudio(MonsterAudioType.MeleeAttackHit);
                    if (particle == null)
                    {
                        particle = particleController.GetParticle(MonsterParticleType.MeleeHit);
                    }
                    particle.transform.position = other.transform.position;
                    particle.Play();
                    SetAndAttack(other.transform);
                }
            }
        }
        private void SetAndAttack(Transform otherTransform)
        {
            CombatPayload payload = new();
            payload.Type = data.combatType;
            payload.Attacker = transform;
            payload.Defender = otherTransform;
            payload.AttackDirection = Vector3.zero;
            payload.AttackStartPosition = transform.position;
            payload.AttackPosition = otherTransform.position;
            payload.StatusEffectName = StatusEffectName.WeakRigidity;
            payload.statusEffectduration = 0.3f;
            payload.Damage = data.damage;
            Attack(payload);
        }
    }
}
```

- MonsterWeaponAttack (내적 계산 공격)

몬스터가 무기를 휘둘러 공격하는 기술로, 무기와 상관 없이 주어진 반지름과 각도를 기준으로 부채꼴 형태의 범위에 내적을 사용하여 플레이어가 부채꼴 안에 있는지 아닌지를 확인하여 공격 여부를 판단한다

```
using System.Collections;
using Assets.Scripts.StatusEffects;
using Channels.Combat;
using UnityEngine;
```

```
namespace Scripts.BehaviourTrees.Monster
{
    public class MonsterWeaponeAttack : MonsterAttackBase
    {
        private WeaponAttackData data;
        private GameObject target;

        protected override void Awake()
        {
            base.Awake();
            target = GameObject.Find("Player");
        }
        public void SetInitialData(WeaponAttackData WeaponData)
        {
            data = WeaponData;
        }
        public override void ExecuteAttack()
        {
            StartCoroutine(AttackFanshape());
        }
        public IEnumerator AttackFanshape()
        {
            float accumTime = 0.0f;
            while (accumTime <= data.duration)
            {
                if (CaculateDotProduct())
                {
                    if (target.CompareTag("Player"))
                    {
                        audioController.PlayAudio(MonsterAudioType.WeaponAttackHit);
                        particleController.PlayParticle(MonsterParticleType.WeaponHit, target.transform);

                        SetAndAttack(data, target.transform);
                        break;
                    }
                }
                accumTime += Time.deltaTime;
                yield return null;
            }
        }
        private bool CaculateDotProduct()
        {
            Vector3 interV = target.transform.position - transform.position;

            float dot = Vector3.Dot(interV.normalized, transform.forward.normalized);
            float theta = Mathf.Acos(dot);
            float degree = Mathf.Rad2Deg * theta;
```

```
            if (degree <= data.angle / 2.0f)
            {
                interV.y = 0;
                if (interV.sqrMagnitude <= data.radius * data.radius)
                {
                    return true;
                }
            }

            return false;
        }
        private void SetAndAttack(WeaponAttackData data, Transform otherTransform)
        {
            CombatPayload payload = new();
            payload.Type = data.combatType;
            payload.Attacker = transform;
            payload.Defender = otherTransform;
            payload.AttackDirection = Vector3.zero;
            payload.AttackStartPosition = transform.position;
            payload.AttackPosition = otherTransform.position;
            payload.StatusEffectName = StatusEffectName.WeakRigidity;
            payload.statusEffectduration = 0.5f;
            payload.Damage = data.damage;
            Attack(payload);
        }
    }
}
```

- MonsterProjectileAttack(투사체 공격)

몬스터의 투사체 공격으로 몬스터에게 투사체 Prefab 을 발사하여 공격하는 방법

```
using Assets.Scripts.Managers;
using Assets.Scripts.StatusEffects;
using Channels.Combat;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class MonsterProjectileAttack : MonsterAttackBase
    {
        private const string path = "Prefabs/Monster/Projectiles/";

        private ProjectileAttackData projectileData;
        private GameObject projectileObj;
        private ProjectileBase projectile;

        protected override void Awake()
        {
            base.Awake();

            string finalPath = path + projectileData.name;
            projectileObj = Instantiate(ResourceManager.Instance.LoadExternResource<GameObject>(finalPath),
transform);
            if (projectileObj == null)
                Debug.LogFormat("{0} Trying To Instanciate {1} But No Prefab On {2} : {0}, {1}", transform.name,
projectileData.name, path);
            else
                projectile = projectileObj.GetComponent<ProjectileBase>();
        }
        public void SetInitialData(ProjectileAttackData projectileData)
        {
            this.projectileData = projectileData;
        }
        public override void ExecuteAttack()
        {
            if (projectileData == null)
                Debug.LogFormat("{0} Has Not Initialized {1} : {0}, {1}" + transform.name, projectileData.name);
            else projectile.Fire();
        }
        public void Explode(Transform transform)
        {
            particleController.PlayParticle(MonsterParticleType.ProjectileHit, transform);
            audioController.PlayAudio(MonsterAudioType.ProjectileHit, transform);
        }
        public void SetAndAttack(Transform otherTransform)
```

```
        {
            CombatPayload payload = new();
            payload.Type = projectileData.combatType;
            payload.Attacker = transform;
            payload.Defender = otherTransform;
            payload.AttackDirection = Vector3.zero;
            payload.AttackStartPosition = transform.position;
            payload.AttackPosition = otherTransform.position;
            payload.StatusEffectName = StatusEffectName.Burn;
            payload.statusEffectduration = 3.0f;
            payload.Damage = (int)projectileData.damage;
            Attack(payload);
        }
    }
}
```

- MonsterFleeSkill (도망가기)

플레이어와 멀리 떨어진 곳으로 도망가는 스킬로 최우선 적으로 몬스터 뒤쪽으로 Ray 를 통해 벽 유무, NavMesh 인 곳인지를 확인하고, 실패 시 랜덤 방향으로 총 5 번의 시도를 한다. 성공 시 빠른 속도로 해당 위치로 이동하게 된다.

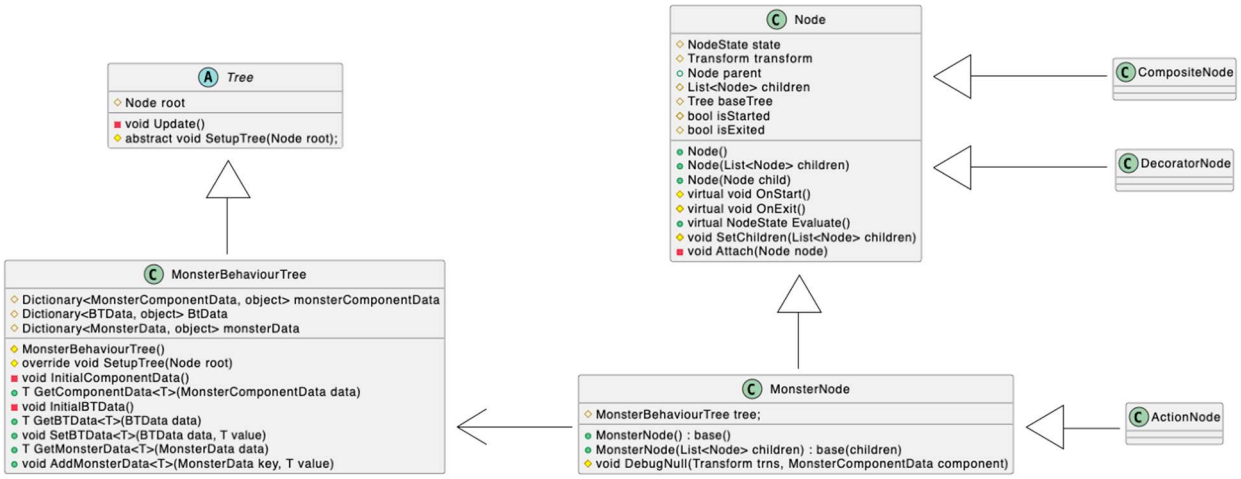
```
using System.Collections;
using UnityEngine;
using UnityEngine.AI;

namespace Scripts.BehaviourTrees.Monster
{
    public class MonsterFleeSkill : MonsterAttackBase
    {
        private const int constMaxAttempts = 5;
        private FleeSkillData fleeData;
        private MonsterStatData statData;
        private NavMeshAgent agent;

        private Vector3 directionVector;
        private Vector3 runAwayVector;
        private Vector3 fleeVector;
        private int attemptFlee;
        private GameObject player;

        protected override void Awake()
        {
            base.Awake();
            player = GameObject.Find("Player");
            agent = GetComponent<NavMeshAgent>();
```

```
}
public void SetInitialData(MonsterStatData statData, FleeSkillData fleeData)
{
    this.statData = statData;
    this.fleeData = fleeData;
}
public override void ExecuteAttack()
{
    if(statData==null||fleeData==null)
    {
        Debug.LogFormat("{0} Has Not Initialized WeaponAttack : {0}, WeaponAttack" + transform.name);
        return;
    }
    attemptFlee = 0;
    directionVector = player.transform.position - transform.position;
    directionVector.y = 0.0f;
    runAwayVector = directionVector.normalized * -fleeData.fleeDistance;
    fleeVector = transform.position + runAwayVector;

    // Check Walls
    RaycastHit hit;
    do
    {
        bool isHittedWall = true;
        if (Physics.Raycast
            (transform.position, runAwayVector.normalized, out hit, fleeData.fleeDistance))
        {
            if (hit.collider.tag == "Wall")
            {
                runAwayVector = Random.onUnitSphere;
            }
            else isHittedWall = false;
        }
        else isHittedWall = false;

        // Check NavMesh
        if (!isHittedWall)
        {
            NavMeshHit navMeshHit;
            if (NavMesh.SamplePosition(fleeVector, out navMeshHit, 1.0f, NavMesh.AllAreas))
            {
                // Found Runaway Point
                agent.destination = navMeshHit.position;
                agent.speed = fleeData.fleeSpeed;
                StartCoroutine(FinishUsingFlee());
                break;
            }
        }
```

```
                attemptFlee++;
            } while (attemptFlee < constMaxAttempts);
        }
        private IEnumerator FinishUsingFlee()
        {
            yield return new WaitForSeconds(fleeData.duration - fleeData.animationHold);
            agent.speed = statData.speed;
        }
    }
}
```

4. Behaviour Tree

4-1. Behaviour Tree & Node



- Tree

MonsterBehaviourTree 의 부모 클래스로 root 노드 설정 추상함수와 Update 마다 root 노드를 Evaluate 한다

```
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public abstract class Tree : MonoBehaviour
    {
        protected Node root = null;

        private void Update()
        {
            if (root != null) root.Evaluate();
        }
        protected abstract void SetupTree(Node root);
    }
}
```

- MonsterBehaviourTree

행동트리에서는 Dictionary 를 통해 3 가지 정보를 관리한다.

1. 몬스터가 가지고 있는 모든 Component 관리
2. 몬스터의 기본 Stat 과 공격 정보 관리
3. 행동 트리에서 필요한 boolean 변수, 플레이어와의 거리 등 행동트리 내에서 사용되고 지속적으로 update 되어야 하는 변수 관리

모든 MonsterNode 는 MonsterBehaviourTree 를 변수로 가지고 있으며, 각 몬스터가 가지고 있는 MonsterNode 는 동일한 MonsterBehaviourTree 를 공유한다.

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

namespace Scripts.BehaviourTrees.Monster
{
    public class MonsterBehaviourTree : Tree
    {
        protected Dictionary<MonsterComponentData, object> monsterComponentData = new();
        protected Dictionary<BTData, object> BtData = new();
        protected Dictionary<MonsterData, object> monsterData;

        protected MonsterBehaviourTree()
        {
            InitialComponentData();
            InitialBTData();
        }
        protected override void SetupTree(Node root)
        {
            this.root = root;
        }
        // >> : ComponentData
        private void InitialComponentData()
        {
            NavMeshAgent agent = GetComponent<NavMeshAgent>();
            monsterComponentData.Add(MonsterComponentData.AGENT, agent);

            MonsterBase monster = GetComponent<MonsterBase>();
            monsterComponentData.Add(MonsterComponentData.MONSTER, monster);

            MonsterParticleController particle = GetComponent<MonsterParticleController>();
            monsterComponentData.Add(MonsterComponentData.PARTICLE, particle);

            AudioSource audioSource = GetComponent<AudioSource>();
            monsterComponentData.Add(MonsterComponentData.AUDIO, audioSource);

            MonsterAudioController audioController = GetComponent<MonsterAudioController>();
            monsterComponentData.Add(MonsterComponentData.AUDIO_CON, audioController);

            Transform transform = GetComponent<Transform>();
            monsterComponentData.Add(MonsterComponentData.TRANSFORM, transform);

            Animator animator = GetComponent<Animator>();
            monsterComponentData.Add(MonsterComponentData.ANIMATOR, animator);
```

```
            Transform patrolPoints = transform.Find("PatrolPoints");
            monsterComponentData.Add(MonsterComponentData.PATROL_POINTS, patrolPoints);

            DetectAI detectPlayer = transform.Find("DetectPlayerAI").GetComponent<DetectAI>();
            monsterComponentData.Add(MonsterComponentData.PlayerDetectAI, detectPlayer);

            DetectAI detectChase = transform.Find("DetectChaseAI").GetComponent<DetectAI>();
            monsterComponentData.Add(MonsterComponentData.ChaseDetectAI, detectChase);

            Vector3[] patorlPoints =
transform.Find("PatrolPoints").GetComponent<PatrolPoints>().GetPatrolPointst();
            monsterComponentData.Add(MonsterComponentData.PATROL_POINTS, patrolPoints);
        }
        public T GetComponentData<T>(MonsterComponentData data)
        {
            object obj = null;
            if (monsterComponentData.TryGetValue(data, out obj))
            {
                if (obj is T) return (T)obj;
            }
            else
            {
                Debug.LogFormat("Trying To Get Type Does Not Match : {0}, {1} " + data.ToString(),
transform.name);
                return default(T);
            }

            Debug.LogFormat(gameObject.name + "Trying To Access Object Does Not Have : {0}, {1}" +
data.ToString(), transform.name);
            return default(T);
        }

        // >> : BTData
        private void InitialBTData()
        {
            SetBTData<bool>(BTData.bOnSpawnPosition, true);
            SetBTData<bool>(BTData.bOvertraveld, false);
            SetBTData<bool>(BTData.bReturning, false);
        }
        public T GetBTData<T>(BTData data)
        {
            object obj;
            if (BtData.TryGetValue(data, out obj))
            {
                if (obj is T) return (T)obj;
                else
                {
```

```
                    Debug.LogFormat("Trying To Get Component Does Not Match : {0}, {1} " + data.ToString(),
transform.name);
                    return default(T);
                }
            }
            else
            {
                T defaultValue = default(T);
                BtData.Add(data, defaultValue);
                return defaultValue;
            }
        }
        public void SetBTData<T>(BTData data, T value)
        {
            if (BtData.ContainsKey(data))
            {
                if (BtData[data] is T)
                {
                    BtData[data] = value;
                    return;
                }
                else
                {
                    Debug.LogFormat("Trying To Get Data Does Not Match : {0}, {1} " + data.ToString(),
transform.name);
                    return;
                }
            }

            Debug.LogFormat("Trying to access data that does not exist: {0}, {1}", data.ToString(), transform.name);
        }

        // >> : MonsterData
        public T GetMonsterData<T>(MonsterData data)
        {
            object obj;
            if (monsterData.TryGetValue(data, out obj))
            {
                if (obj is T) return (T)obj;
                else
                {
                    Debug.LogFormat("Trying To Get Monster Data Does Not Match : {0}, {1} " + data.ToString(),
transform.name);
                    return default(T);
                }
            }

            Debug.LogFormat(gameObject.name + "Trying To Access Monster Data Does Not Have : {0}, {1}" +
data.ToString(), transform.name);
```

```
            return default(T);
        }
        public void AddMonsterData<T>(MonsterData key, T value)
        {
            if(monsterData.ContainsKey(key))
            {
                Debug.LogFormat("{0} Is Trying To Add Key {1} Already Exists : {0}, {1}", transform.name,
key.ToString());
                return;
            }
            monsterData.Add(key, value);
        }
    }
}
```

- Node

MonsterNode, Composite Node, DecoratorNode 의 부모 클래스로 ChildNode 관리

```
using System.Collections.Generic;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public enum NodeState
    {
        RUNNING,
        SUCCESS,
        FAILURE,
    }

    public class Node
    {
        protected NodeState state;
        protected Transform transform;
        public Node parent;
        protected List<Node> children;
        protected Tree baseTree;
        protected bool isStarted;
        protected bool isExited;

        public Node()
        {
            parent = null;
        }
        public Node(List<Node> children)
        {
            foreach (Node child in children)
```

```
                Attach(child);
        }
        public Node(Node child)
        {
            Attach(child);
        }

        protected virtual void OnStart() { }
        protected virtual void OnExit() { }
        public virtual NodeState Evaluate() => NodeState.FAILURE;

        protected void SetChildren(List<Node> children)
        {
            foreach (Node child in children)
                Attach(child);
        }

        private void Attach(Node node)
        {
            node.parent = this;
            children.Add(node);
        }
    }
}
```

- MonsterNode

MonsterBehaviourTree 를 가지고 있는 Monster 에서 사용되는 Node

```
using System.Collections.Generic;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class MonsterNode : Node
    {
        protected MonsterBehaviourTree tree;

        public MonsterNode() : base()
        {
            tree = baseTree as MonsterBehaviourTree;
            if (tree == null)
                Debug.LogFormat("Tree DownCasting To MonsterBehaviourTree Failed : {0}", transform.name);
            transform = tree.GetComponentData<Transform>(MonsterComponentData.TRANSFORM);
        }
        public MonsterNode(List<Node> children) : base(children) { }

        protected void DebugNull(Transform trns, MonsterComponentData component)
        {
```

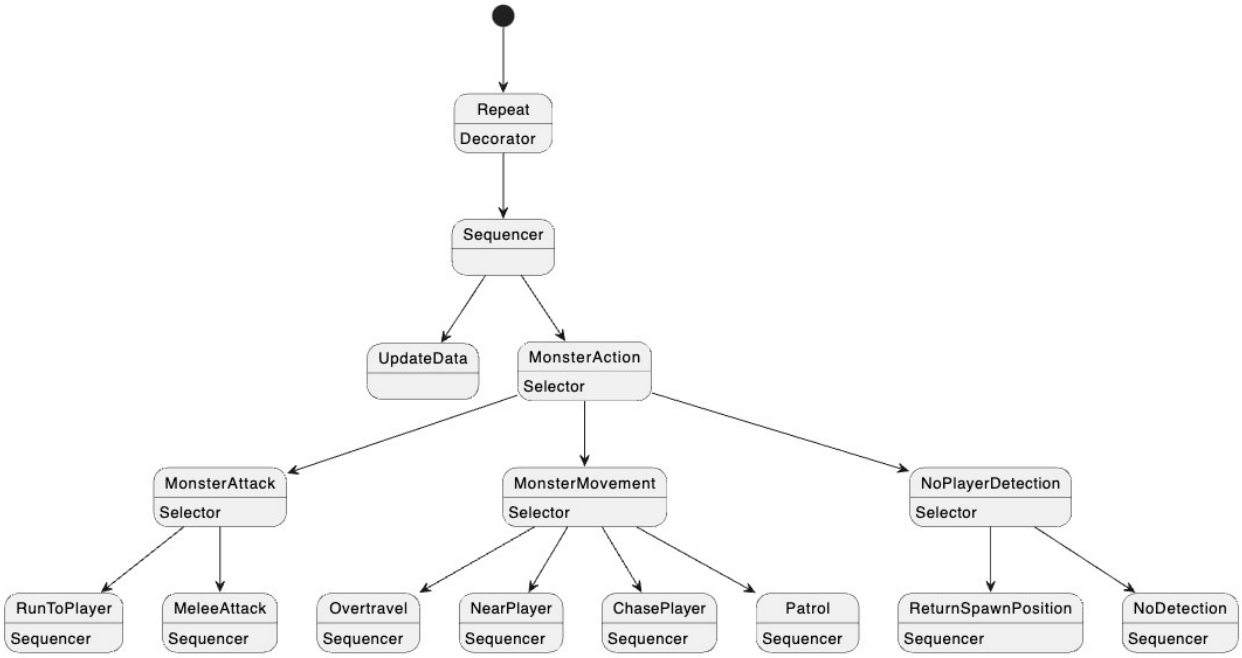
```
                Debug.LogFormat("{0} Is Trying To Access {1}, Which It Does Not Have : {0} / {1}", trns.name,
component.ToString());
            }
        }
}
```

4-2. Monster Behaviour Tree

각 몬스터의 BehaviourTree 로 만들어진 Sequence 들을 필요에 따라 추가한다. 몬스터의 기본 우선순위 행동은 공격->움직임->플레이어 감지 순으로 행동한다.

4-2.1 평범한 스켈레톤

- 설계



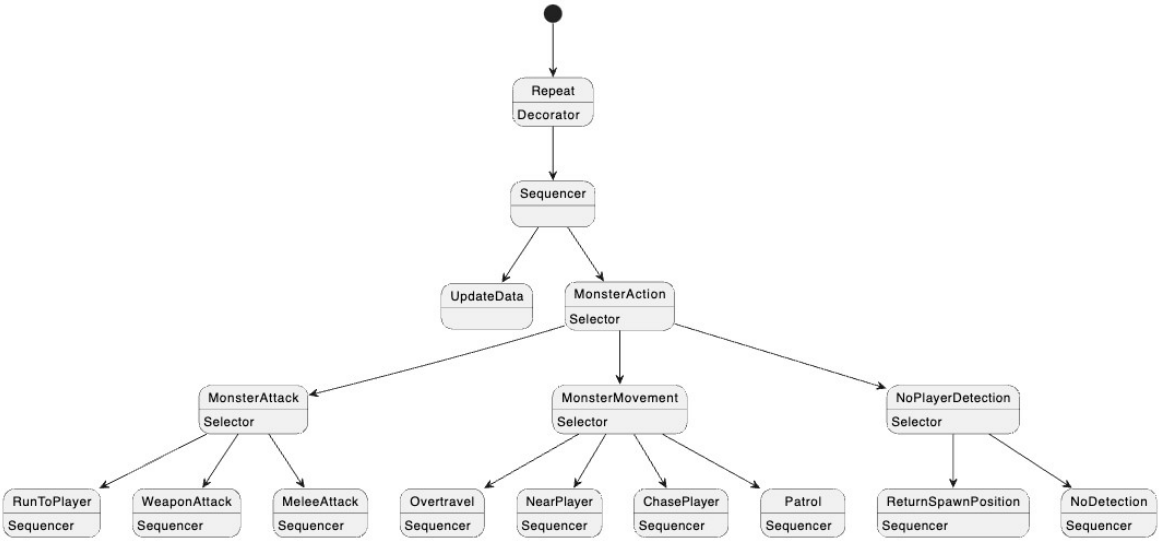
- 코드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class NormalSkeletonBT : MonsterBehaviourTree
    {
        private void Start()
        {
            // Monster Attack Sequences
            Select attackSelect = new(new List<Node>
            {
                new SeqRunToPlayer(),
```

```
                new SeqMeleeAttack()
            });

            // Monster Cant Attack, But Can Move Sequences
            Select movementSelect = new(new List<Node>
            {
                new SeqOvertravel(),
                new SeqNearPlayer(),
                new SeqChasePlayer(),
                new SeqPatrol(),
            });

            // Monster Cant detect player Sequences
            Select noDetectSelect = new(new List<Node>
            {
                new SeqReturnSpawnPosition(),
                new SeqNoDetection(),
            });

            // Combine All Sequences
            Select allSequence = new(new List<Node>
            {
                attackSelect,
                movementSelect,
                noDetectSelect,
            });

            // Combine Update Data
            Sequence normalSkeletonNodes = new(new List<Node>
            {
                new ActionUpdateData(),
                allSequence,
            });

            SetupTree(new Repeater(normalSkeletonNodes));
        }
    }
}
```

4-2.2 모험가 스켈레톤

- 설계



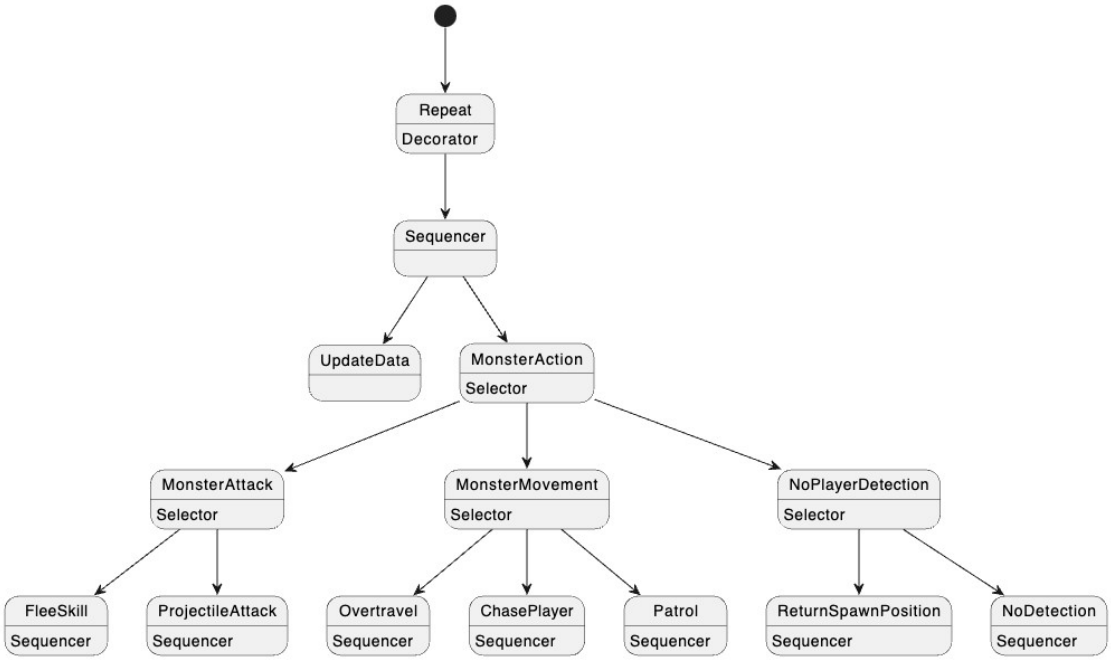
- 코드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class AdventureSkeletonBT : MonsterBehaviourTree
    {
        private void Start()
        {
            Select attackSelect = new(new List<Node>
            {
                new SeqRunToPlayer(),
                new SeqWeaponAttack(),
                new SeqMeleeAttack()
            });
            Select movementSelect = new(new List<Node>
            {
                new SeqOvertravel(),
                new SeqNearPlayer(),
                new SeqChasePlayer(),
                new SeqPatrol(),
            });
            Select noDetectSelect = new(new List<Node>
            {
                new SeqReturnSpawnPosition(),
                new SeqNoDetection(),
            });

            Select allSequence = new(new List<Node>
```

```
            {
                attackSelect,
                movementSelect,
                noDetectSelect,
            });

            Sequence adventureSkeletonNodes = new(new List<Node>
            {
                new ActionUpdateData(),
                allSequence,
            });

            SetupTree(new Repeater(adventureSkeletonNodes));
        }
    }
}
```

4-2.3 마법사 스켈레톤

- 설계



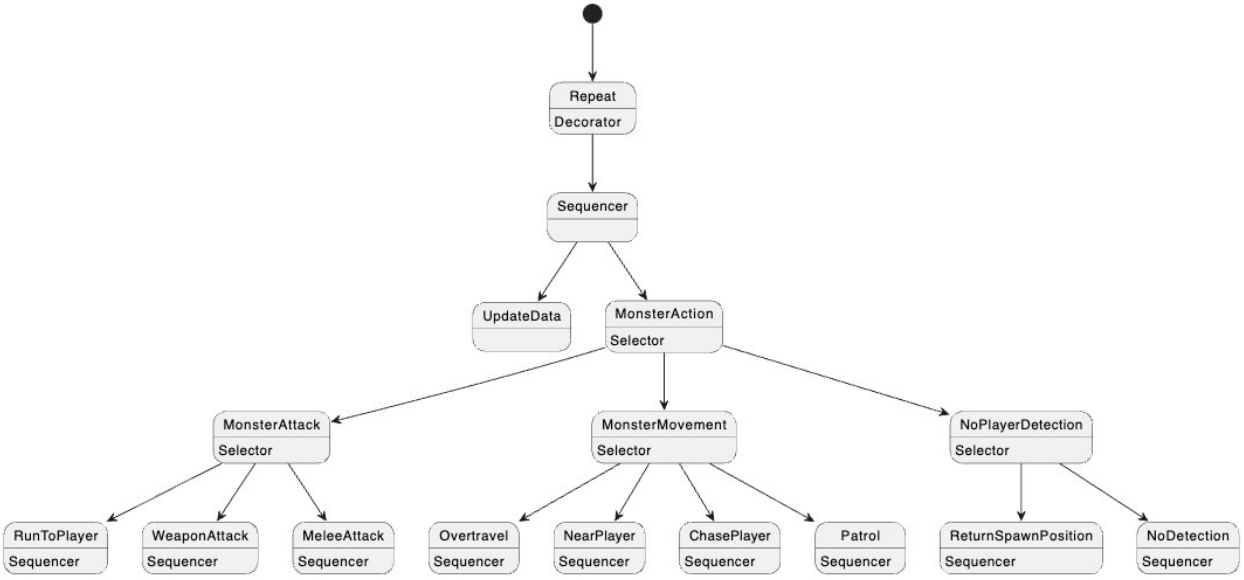
- 코드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class WizardSkeletonBT : MonsterBehaviourTree
    {
        private void Start()
        {
```

```
            Select attackSelect = new(new List<Node>
            {
                new SeqFleeSkill(),
                new SeqProjectileAttack(),
            });
            Select movementSelect = new(new List<Node>
            {
                new SeqOvertravel(),
                new SeqChasePlayer(),
                new SeqPatrol(),
            });
            Select noDetectSelect = new(new List<Node>
            {
                new SeqReturnSpawnPosition(),
                new SeqNoDetection(),
            });

            Select allSequence = new(new List<Node>
            {
                attackSelect,
                movementSelect,
                noDetectSelect,
            });

            Sequence wizardSkeletonNodes = new(new List<Node>
            {
                new ActionUpdateData(),
                allSequence,
            });

            SetupTree(new Repeater(wizardSkeletonNodes));
        }
    }
}
```

4-2.4 공대원 스켈레톤

- 설계



- 코드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class GuildGuardSkeletonBT : MonsterBehaviourTree
    {
        private void Start()
        {
            Select attackSelect = new(new List<Node>
            {
                new SeqRunToPlayer(),
                new SeqWeaponAttack(),
                new SeqMeleeAttack()
            });
            Select movementSelect = new(new List<Node>
            {
                new SeqOvertravel(),
                new SeqNearPlayer(),
                new SeqChasePlayer(),
                new SeqPatrol(),
            });
            Select noDetectSelect = new(new List<Node>
            {
                new SeqReturnSpawnPosition(),
                new SeqNoDetection(),
            });
```

```
                Sequence allSequence = new(new List<Node>
                {
                    attackSelect,
                    movementSelect,
                    noDetectSelect,
                });

                Sequence guildguardSkeletonNodes = new(new List<Node>
                {
                    new ActionUpdateData(),
                    allSequence,
                });

                SetupTree(new Repeater(guildguardSkeletonNodes));
            }
        }
}
```

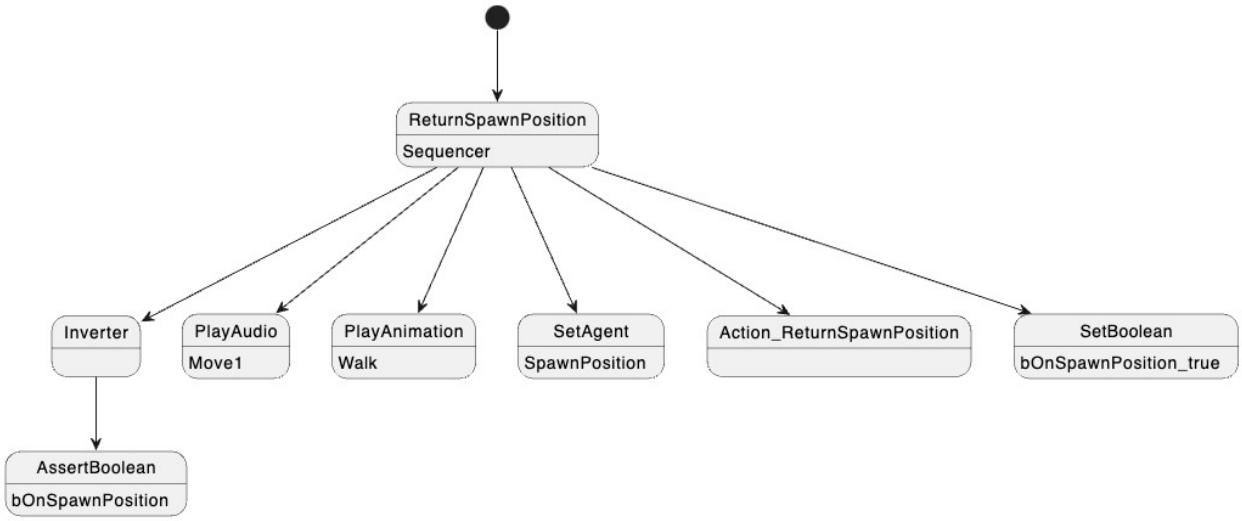
4-3. Sequence Behaviour Tree

4-3.1 플레이어 미감지

- ReturnSpawnPosition

플레이어가 감지된 상태에서 다시 감지하지 못할 때 자신의 Spawn 위치로 돌아가는 Sequence

설계



코드

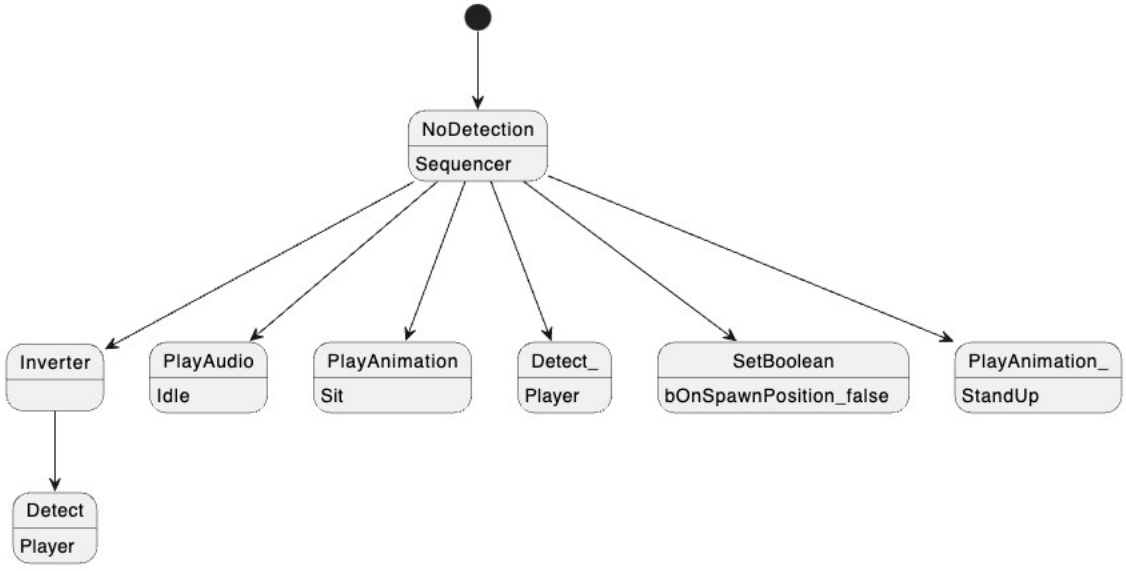
```
using System.Collections.Generic;
using UnityEngine;
```

```
namespace Scripts.BehaviourTrees.Monster
{
    public class SeqReturnSpawnPosition : Sequence
    {
        public SeqReturnSpawnPosition()
        {
            List<Node> children = new()
            {
                new Inverter(new ActionAssertBoolean(tree.GetBTData<bool>(BTData.bOnSpawnPosition))),
                new ActionPlayAudio(MonsterAudioType.Move1, true, true),
                new ActionPlayAnimation(AnimationType.WALK),
                new ActionSetAgent(tree.GetMonsterData<Vector3>(MonsterData.v3SpawnPosition)),
                new ActionReturnSpawnPosition(),
                new ActionSetBoolean(BTData.bOnSpawnPosition, true),
            };

            SetChildren(children);
        }
    }
}
```

- NoDetection

플레이어를 감지하지 못한 상태에서 대기와 대기 중 플레이어 감지 했을 때의 행동 Sequence

설계



코드

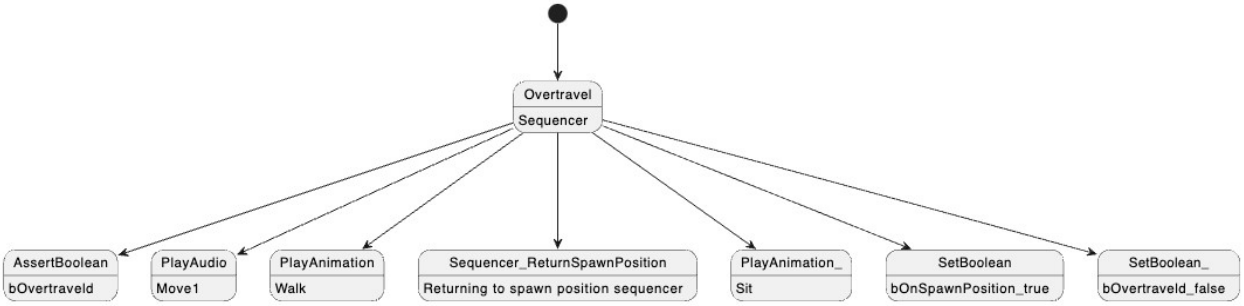
```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqNoDetection : Sequence
    {
        public SeqNoDetection()
        {
            List<Node> children = new()
            {
                // undetected player
                new Inverter(new ActionDetect(DetectType.PLAYER)),
                new ActionPlayAudio(MonsterAudioType.Idle,false, true),
                new ActionPlayAnimation(AnimationType.SIT, true),
                new ActionDetect(DetectType.PLAYER),
                // if detected player
                new ActionSetBoolean(BTData.bOnSpawnPosition, false),
                new ActionPlayAnimation(AnimationType.STANDUP, true),
            };

            SetChildren(children);
        }
    }
}
```

4-3.2 움직임

- Overtravel

몬스터가 spawn 위치 기준으로 허용된 범위 밖으로 나갈 때 자신의 Spawn 위치로 돌아오는 Sequence

설계



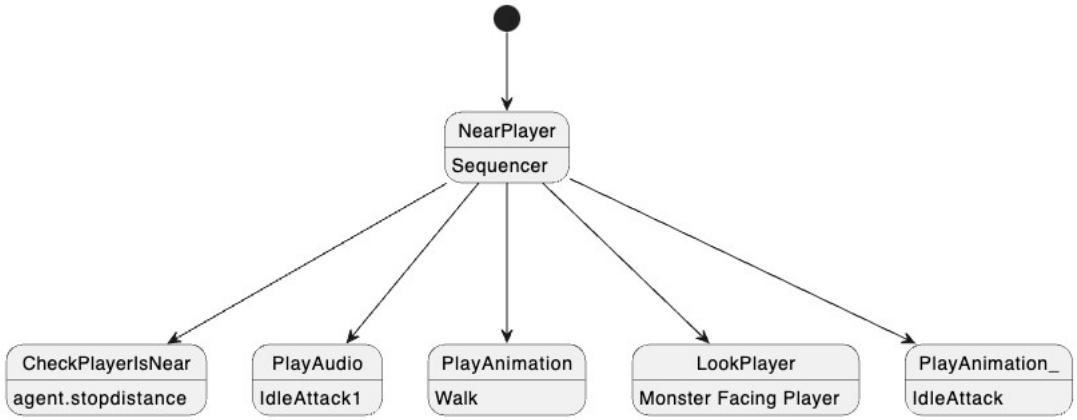
코드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqOvertravel : Sequence
    {
        public SeqOvertravel()
        {
            List<Node> children = new()
            {
                new ActionAssertBoolean(tree.GetBTData<bool>(BTData.bOvertraveld)),
                new ActionPlayAudio(MonsterAudioType.Move1),
                new ActionPlayAnimation(AnimationType.WALK),
                new SeqReturnSpawnPosition(),
                new ActionPlayAnimation(AnimationType.SIT),
                new ActionSetBoolean(BTData.bOnSpawnPosition, true),
                new ActionSetBoolean(BTData.bOvertraveld, false),
            };

            SetChildren(children);
        }
    }
}
```

- NearPlayer

설정된 범위 안에 플레이어가 있을 때 플레이어를 바라보게 돌아주는 Sequence

설계



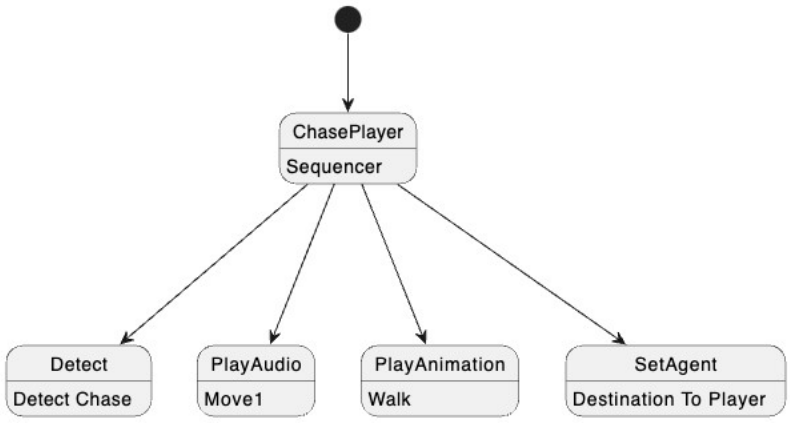
코드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqNearPlayer : Sequence
    {
        public SeqNearPlayer()
        {
            List<Node> children = new()
            {
                new ActionCheckPlayerIsNear(),
                new ActionPlayAudio(MonsterAudioType.IdleAttack1),
                new ActionPlayAnimation(AnimationType.WALK),
                new ActionLookPlayer(),
                new ActionPlayAnimation(AnimationType.IDLE_ATTACK),
            };

            SetChildren(children);
        }
    }
}
```

- ChasePlayer

플레이어를 쫓아가는 Sequence

설계



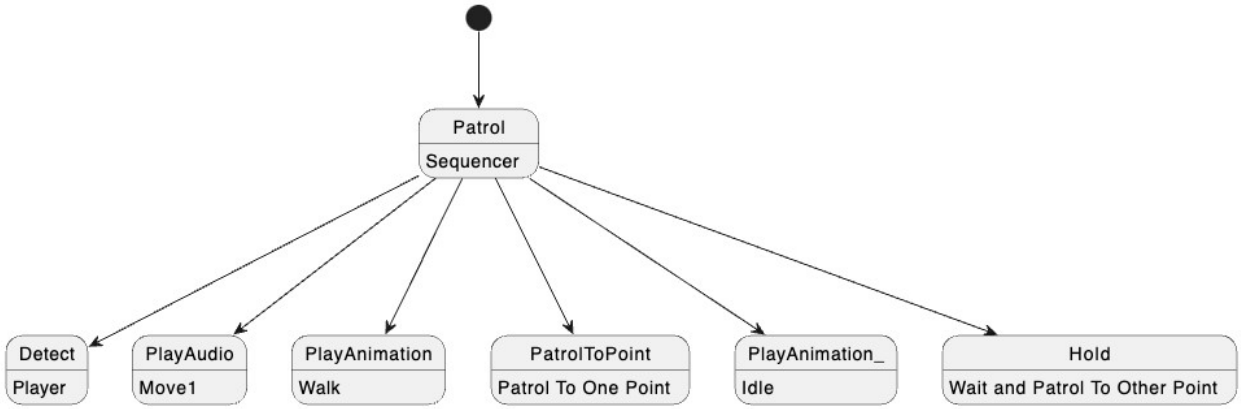
코드

```
using System.Collections.Generic;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqChasePlayer : Sequence
    {
        public SeqChasePlayer()
        {
            List<Node> children = new()
            {
            new ActionDetect(DetectType.CHASE),
            new ActionPlayAudio(MonsterAudioType.Move1),
            new ActionPlayAnimation(AnimationType.WALK),
            new ActionSetAgent(tree.GetBTData<Vector3>(BTData.v3PlayerPosition)),
            };

            SetChildren(children);
        }

    }
}
```

- Patrol

플레이어가 감지 되었으나 쫓아가는 감지 범위 밖에 있을 때 미리 설정된 Point 들을 정찰하는 Sequence

설계



코드

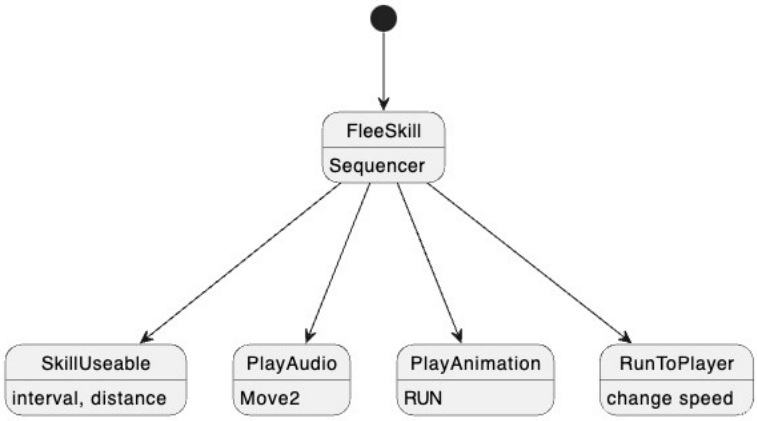
```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqPatrol : Sequence
    {
        public SeqPatrol()
        {
            List<Node> children = new()
            {
                new ActionDetect(DetectType.PLAYER),
                new ActionPlayAudio(MonsterAudioType.Move1),
                new ActionPlayAnimation(AnimationType.WALK),
                new ActionPatrolToPoint(),
                new ActionPlayAnimation(AnimationType.IDLE),
                new ActionHold(1.0f),
            };

            SetChildren(children);
        }

    }
}
```

4-3.3 공격

몬스터의 공격 가능 여부 (플레이어와의 거리, 공격의 Interval)을 확인 후 공격 실행한다. 공격 실행은 해당 공격의 ExecuteAttack 함수를 호출하여 행동트리가 아닌 몬스터 공격 script 에서 실행된다. 행동트리는 해당 공격의 Audio 와 Animation 을 필요할 때 실행시켜 준다.

- RunToPlayer

설계

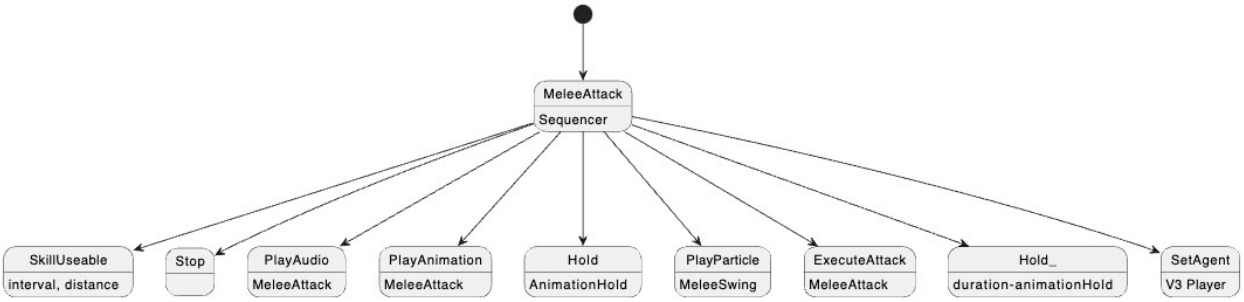


코드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqRunToPlayer : Sequence
    {
        public SeqRunToPlayer()
        {
            RunToPlayerData runToPlayer = tree.GetMonsterData<RunToPlayerData>(MonsterData.RunToPlayer);

            List<Node> children = new()
            {
                new ActionSkillUseable(runToPlayer.interval, runToPlayer.attackableDistance),
                new ActionPlayAudio(MonsterAudioType.MoveSkill),
                new ActionPlayAnimation(AnimationType.RUN),
                new ActionRunToPlayer()
            };

            SetChildren(children);
        }
    }
}
```

- MeleeAttack

설계

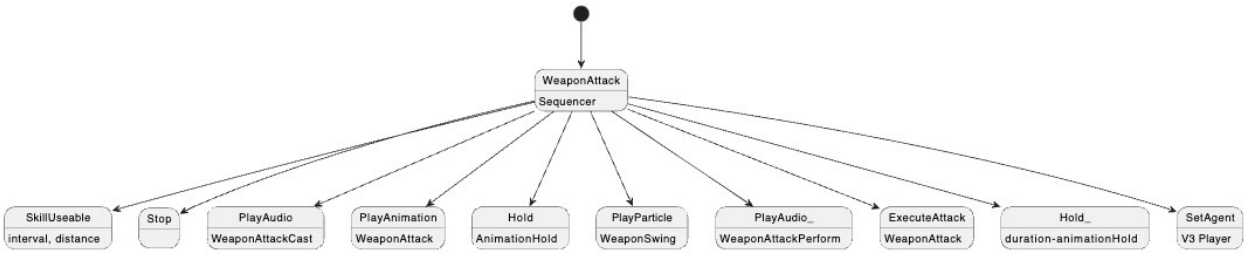


코드

```
using System.Collections.Generic;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqMeleeAttack : Sequence
    {
        public SeqMeleeAttack()
        {
            MeleeAttackData meleeAttack = tree.GetMonsterData<MeleeAttackData>(MonsterData.Melee);

            List<Node> children = new()
            {
                new ActionSkillUseable(meleeAttack.interval, meleeAttack.attackableDistance),
                new ActionStop(),
                new ActionPlayAudio(MonsterAudioType.MeleeAttack,true, false),
                new ActionPlayAnimation(AnimationType.MELEE),
                new ActionHold(meleeAttack.animationHold),
                new ActionPlayParticle(MonsterParticleType.MeleeAttack),
                new ActionExecuteAttack(MonsterSkill.Melee),
                new ActionHold(meleeAttack.duration-meleeAttack.animationHold),
                new ActionSetAgent(tree.GetBTData<Vector3>(BTData.v3PlayerPosition)),
            };

            SetChildren(children);
        }
    }
}
```

- WeaponAttack

설계



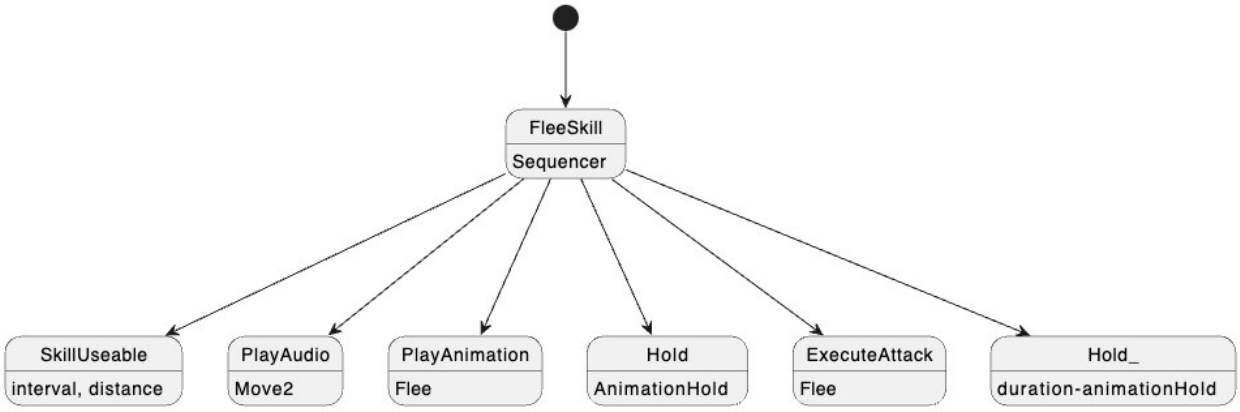
코드

```
using System.Collections.Generic;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqWeaponAttack : Sequence
    {
        public SeqWeaponAttack()
        {
            WeaponAttackData weaponData =
tree.GetMonsterData<WeaponAttackData>(MonsterData.Weapon);

            List<Node> children = new()
            {
                new ActionSkillUseable(weaponData.interval,
weaponData.attackableDistance),
                new ActionStop(),
                new ActionPlayAudio(MonsterAudioType.WeaponAttackCast),
                new ActionPlayAnimation(AnimationType.WEAPON),
                new ActionHold(weaponData.animationHold),
                new ActionPlayParticle(MonsterParticleType.WeaponSwing),
                new ActionPlayAudio(MonsterAudioType.WeaponAttackPerform),
                new ActionExecuteAttack(MonsterSkill.Weapon),
                new ActionHold(weaponData.duration-weaponData.animationHold),
                new ActionSetAgent(tree.GetBTData<Vector3>(BTData.v3PlayerPosition)),
            };

            SetChildren(children);
        }
```

```
    }
}
```

- FleeSkill

설계

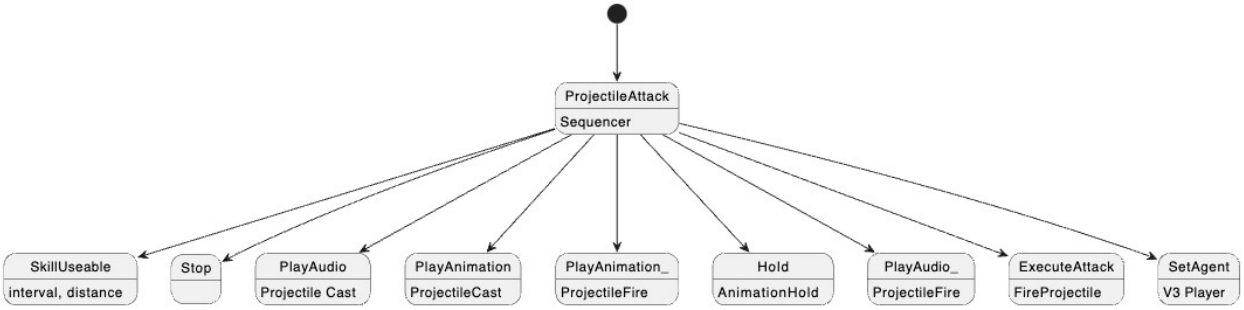


코드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqFleeSkill : Sequence
    {
        public SeqFleeSkill()
        {
            FleeSkillData fleeData = tree.GetMonsterData<FleeSkillData>(MonsterData.Flee);

            List<Node> children = new()
            {
                new ActionSkillUseable(fleeData.interval, fleeData.attackableDistance),
                new ActionPlayAudio(MonsterAudioType.Move2),
                new ActionPlayAnimation(AnimationType.FLEE),
                new ActionHold(fleeData.animationHold),
                new ActionExecuteAttack(MonsterSkill.Flee),
                new ActionHold(fleeData.duration-fleeData.animationHold),
            };

            SetChildren(children);
        }
    }
}
```

- ProjectileAttack

설계



코드

```
using System.Collections.Generic;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class SeqProjectileAttack : Sequence
    {
        public SeqProjectileAttack()
        {
            ProjectileAttackData data = tree.GetMonsterData<ProjectileAttackData>(MonsterData.Projectile);

            List<Node> children = new()
            {
                new ActionSkillUseable(data.interval, data.attackableDistance),
                new ActionStop(),
                new ActionPlayAudio(MonsterAudioType.ProjectileAttack, true, false),
                new ActionPlayAnimation(AnimationType.PROJECTILE_CAST,true),
                new ActionPlayAnimation(AnimationType.PROJECTILE_FIRE, false),
                new ActionHold(data.animationHold),
                new ActionPlayAudio(MonsterAudioType.ProjectileFire),
                new ActionExecuteAttack(MonsterSkill.Projectile),
                new ActionSetAgent(tree.GetBTData<Vector3>(BTData.v3PlayerPosition)),
            };

            SetChildren(children);
        }
    }
}
```

4-4. ActionNode

4-4.1 공통

- ActionAssertBoolean

Boolean 이 true 면 성공, fail 이면 실패를 반환하며 반대로 사용하는 방법은 Invertal 과 함께 사용한다

```
namespace Scripts.BehaviourTrees.Monster
{
    public class ActionAssertBoolean : MonsterNode
    {
        private bool boolean;
        public ActionAssertBoolean(bool boolean)
        {
            this.boolean = boolean;
        }
        public override NodeState Evaluate()
        {
            if (boolean) return NodeState.SUCCESS;
            else return NodeState.FAILURE;
        }
    }
}
```

- ActionDetect

몬스터가 플레이어를 인지, 추적 감지 여부를 판단하여 성공, 실패를 반환한다.

```
namespace Scripts.BehaviourTrees.Monster
{
    public class ActionDetect : MonsterNode
    {
        private DetectType type;

        public ActionDetect(DetectType type)
        {
            this.type = type;
        }
        public override NodeState Evaluate()
        {
            switch (type)
            {
                case DetectType.PLAYER:
                    if (tree.GetComponentData<DetectAI>(MonsterComponentData.PlayerDetectAI).IsDetected)
                        return NodeState.SUCCESS;
                    break;
                case DetectType.CHASE:
                    if (tree.GetComponentData<DetectAI>(MonsterComponentData.ChaseDetectAI).IsDetected)
```

```
                    return NodeState.SUCCESS;
                break;
            }

            return NodeState.FAILURE;
        }
    }
}
```

- ActionHold

몬스터가 다음 ActionNode 를 실행하기 전 대기해야 할 때 사용된다.

```
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionHold : MonsterNode
    {
        private float holdTime;
        private float accumTime=0;
        public ActionHold(float holdTime= 1.0f) : base()
        {
            this.holdTime = holdTime;
        }
        protected override void OnStart()
        {
            accumTime = 0;
        }
        public override NodeState Evaluate()
        {
            if (accumTime >= holdTime) return NodeState.SUCCESS;
            accumTime += Time.deltaTime;
            return NodeState.RUNNING;
        }
    }
}
```

- ActionPlayAnimation

몬스터의 Animation 을 실행해주는 ActionNode 로 Animation 을 실행하고 바로 다음 ActionNode 로 넘어갈지, 아니면 해당 Animation 이 끝날 때 까지 기다릴지 설정해줄 수 있다.

```
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionPlayAnimation : MonsterNode
```

```
    {
        Animator animator;
        private AnimationType animation;
        private bool waitToEnd;
        private float animationLength;
        private float accumTime;

        public ActionPlayAnimation(AnimationType animation, bool waitToEnd = false)
        {
            if(animator==null)
                animator = tree.GetComponentData<Animator>(MonsterComponentData.ANIMATOR);
            if (animator == null)
                DebugNull(transform, MonsterComponentData.ANIMATOR);

            this.animation = animation;
            this.waitToEnd = waitToEnd;
        }
        protected override void OnStart()
        {
            animator.SetTrigger(animation.ToString());

            if (waitToEnd)
            {
                accumTime = 0;
                AnimatorClipInfo[] clipInfo = animator.GetCurrentAnimatorClipInfo(0);
                animationLength = clipInfo[0].clip.length;
            }
        }
        public override NodeState Evaluate()
        {
            if(waitToEnd)
            {
                if (accumTime <= animationLength)
                {
                    accumTime += Time.deltaTime;
                    return NodeState.RUNNING;
                }
                else return NodeState.SUCCESS;
            }

            return NodeState.SUCCESS;
        }
    }
}
```

- ActionPlayAudio

몬스터의 Audio 를 실행해주는 ActionNode 로 ScriptableObject 에 준비된 Audio 를 찾아 실행해준다. 다른 Audio 가 실행 중일 때 중단하고 실행할 것인지, 반복재생해줄 것인지 설정해줄 수 있다

```
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionPlayAudio : MonsterNode
    {
        private AudioSource audioSource;
        private MonsterAudioController audioController;
        private MonsterAudioType audioType;
        private bool isLoop;
        private bool isInteruptable;

        public ActionPlayAudio(MonsterAudioType audioType, bool isInteruptable = true, bool isLoop = true) :
base()
        {
            if (audioSource == null)
                audioSource = tree.GetComponentData<AudioSource>(MonsterComponentData.AUDIO);
            if (audioSource == null)
                DebugNull(transform, MonsterComponentData.AUDIO);

            if (audioController == null)
                audioController =
tree.GetComponentData<MonsterAudioController>(MonsterComponentData.AUDIO_CON);
            if (audioController == null)
                DebugNull(transform, MonsterComponentData.AUDIO_CON);

            this.audioType = audioType;
            this.isLoop = isLoop;
            this.isInteruptable = isInteruptable;
        }

        public override NodeState Evaluate()
        {
            if (isLoop)
                audioSource.loop = true;
            else audioSource.loop = false;

            if (!isInteruptable)
            {
                if (audioSource.isPlaying)
                    return NodeState.SUCCESS;
            }
```

```
                AudioClip clip = audioController.GetAudio(audioType);
                if(clip==null)
                {
                    Debug.Log(transform.name + "Try To Play Audio Does Not Have: " + audioType.ToString());
                    return NodeState.FAILURE;
                }
                audioSource.clip = clip;
                audioSource.Play();

                return NodeState.SUCCESS;
            }
        }
}
```

- ActionPlayParticle

Scriptable Object 로 준비된 Particle 을 실행시켜주는 ActionNode

```
namespace Scripts.BehaviourTrees.Monster
{
    public class ActionPlayParticle : MonsterNode
    {
        private MonsterParticleController particleController;
        private MonsterParticleType particleType;

        public ActionPlayParticle(MonsterParticleType particleType)
        {
            particleController =
tree.GetComponentData<MonsterParticleController>(MonsterComponentData.PARTICLE);
            this.particleType = particleType;
        }
        public override NodeState Evaluate()
        {
            particleController.PlayParticle(particleType);

            return NodeState.SUCCESS;
        }
    }
}
```

- ActionSetAgent

몬스터의 NavMeshAgent 의 목표 지점을 설정해주는 Node

```
using UnityEngine;
using UnityEngine.AI;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionSetAgent : MonsterNode
```

```
    {
        private NavMeshAgent agent;
        private Vector3 destination;

        public ActionSetAgent(Vector3 destination)
        {
            agent = tree.GetComponentData<NavMeshAgent>(MonsterComponentData.AGENT);
            if (agent == null)
                DebugNull(transform, MonsterComponentData.AGENT);

            this.destination = destination;
        }
        public override NodeState Evaluate()
        {
            agent.destination = destination;
            return NodeState.SUCCESS;
        }
    }
}
```

- ActionSetBoolean

MonsterBehaviourTree 에 있는 Tree 에서 사용되는 정보들 중 Boolean 을 설정할 수 있는 Node

```
namespace Scripts.BehaviourTrees.Monster
{
    public class ActionSetBoolean : MonsterNode
    {
        private BTData data;
        private bool boolean;

        public ActionSetBoolean(BTData data, bool boolean)
        {
            this.data = data;
            this.boolean = boolean;
        }
        public override NodeState Evaluate()
        {
            tree.SetBTData<bool>(data, boolean);
            return NodeState.SUCCESS;
        }
    }
}
```

- ActionStop

몬스터가 공격 등 움직임 말고 다른 행동을 실행할 때 몬스터가 자리에 멈추게 하는 Node

```
using UnityEngine.AI;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionStop : MonsterNode
    {
        private NavMeshAgent agent;

        public ActionStop()
        {
            agent = tree.GetComponentData<NavMeshAgent>(MonsterComponentData.AGENT);
        }
        public override NodeState Evaluate()
        {
            agent.destination = transform.position;
            return NodeState.SUCCESS;
        }
    }
}
```

- ActionUpdateData

지속적으로 Update 되어야 하는 정보들을 관리하는 Node

```
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionUpdateData : MonsterNode
    {
        public ActionUpdateData() { }

        public override NodeState Evaluate()
        {
            Vector3 playerPosition =
tree.GetMonsterData<GameObject>(MonsterData.v3SpawnPosition).transform.position;
            tree.SetBTData<Vector3>(BTData.v3PlayerPosition, playerPosition);

            float playerDistance = Vector3.SqrMagnitude(playerPosition - transform.position);
            tree.SetBTData<float>(BTData.fPlayerDistanceSqr, playerDistance);

            float spawnDistance =
Vector3.SqrMagnitude(tree.GetMonsterData<Vector3>(MonsterData.v3SpawnPosition));
            tree.SetBTData<float>(BTData.fSpawnDistanceSqr, spawnDistance);
```

```
            if (spawnDistance > tree.GetMonsterData<MonsterStatData>(MonsterData.MonsterStat).overtravelDist)
                tree.SetBTData<bool>(BTData.bOvertraveld, true);

            return NodeState.SUCCESS;
        }
    }
}
```

4-4.2 플레이어 미감지

- ActionReturnSpawnPosition

몬스터가 Spawn 된 위치로 복귀를 완료 하였는지 확인하는 Node

```
namespace Scripts.BehaviourTrees.Monster
{
    public class ActionReturnSpawnPosition : MonsterNode
    {
        public override NodeState Evaluate()
        {
            float distance = tree.GetBTData<float>(BTData.fSpawnDistanceSqr);

            if (distance > 0.5f)
                return NodeState.SUCCESS;

            else return NodeState.RUNNING;
        }
    }
}
```

4-4.3 움직임

- ActionCheckPlayerIsNear

플레이어가 몬스터의 설정된 범위 안에 들어왔는지 확인하는 Node

```
namespace Scripts.BehaviourTrees.Monster
{
    public class ActionCheckPlayerIsNear : MonsterNode
    {
        public override NodeState Evaluate()
        {
            float playerDistance = tree.GetBTData<float>(BTData.fPlayerDistanceSqr);
            float playerStopDistance =
tree.GetMonsterData<MonsterStatData>(MonsterData.MonsterStat).stopDistance;

            if (playerDistance < playerStopDistance)
                return NodeState.SUCCESS;

            return NodeState.FAILURE;
        }
```

```
    }
}
```

- ActionLookPlayer

몬스터가 플레이어를 향해 바라보는 Node

```
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionLookPlayer : MonsterNode
    {
        private float accumTime;
        private const float angle = 90.0f;
        private const float radius = 5.0f;

        public override NodeState Evaluate()
        {
            Vector3 playerPos = tree.GetBTData<Vector3>(BTData.v3PlayerPosition);
            Vector3 monsterPos = transform.position;
            Vector3 interV = playerPos - monsterPos;

            float dot = Vector3.Dot(interV.normalized, transform.forward.normalized);
            float theta = Mathf.Acos(dot);
            float degree = Mathf.Rad2Deg * theta;

            if (degree <= angle / 2.0f)
            {
                interV.y = 0;
                if (interV.sqrMagnitude <= radius * radius)
                    return NodeState.SUCCESS;
            }
            else
            {
                Quaternion targetRotation;
                Vector3 directionVector = interV;
                directionVector.y = 0;
                directionVector.Normalize();
                targetRotation = Quaternion.LookRotation(directionVector, Vector3.up);

                transform.rotation= Quaternion.Slerp(transform.rotation, targetRotation, Time.deltaTime * 90.0f);
                accumTime += Time.deltaTime;

                if (accumTime >= 0.5f) return NodeState.FAILURE;
            }

            return NodeState.RUNNING;
        }
```

```
    }
}
```

- ActionPatrolToPoints

미리 설정된 Point 들에 몬스터가 차례로 방문하며 정찰하는 Node

```
using UnityEngine;
using UnityEngine.AI;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionPatrolToPoint : MonsterNode
    {
        NavMeshAgent agent;
        Vector3[] patrolPoints;
        private int patrolNum;
        private int patrolQuant;

        public ActionPatrolToPoint()
        {
            if (patrolPoints == null)
                patrolPoints =
tree.GetComponentData<Vector3[]>(MonsterComponentData.PATROL_POINTS);
            if (patrolPoints == null)
                DebugNull(transform, MonsterComponentData.PATROL_POINTS);

            if (agent == null)
                agent =
tree.GetComponentData<NavMeshAgent>(MonsterComponentData.AGENT);
            if (agent == null)
                DebugNull(transform, MonsterComponentData.AGENT);

            patrolQuant = patrolPoints.Length;
            patrolNum = 0;
        }
        public override NodeState Evaluate()
        {
            agent.destination = patrolPoints[patrolNum];
            float distance = Vector3.SqrMagnitude(transform.position -
patrolPoints[patrolNum]);
```

```
                if(distance<0.1f)
                {
                    patrolNum++;
                    if (patrolNum == patrolQuant)
                        patrolNum = 0;

                    return NodeState.SUCCESS;
                }

                return NodeState.RUNNING;
            }

        }
}
```

4-4.4 공격

- ActionExecuteAttack

몬스터가 가지고 있는 공격을 실행시켜주는 Node

```
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionExecuteAttack : MonsterNode
    {
        MonsterSkill monsterSkill;
        MonsterBase monster;

        public ActionExecuteAttack(MonsterSkill monsterSkill)
        {
            if(monster==null)
            monster = tree.GetComponentData<MonsterBase>(MonsterComponentData.MONSTER);
            if (monster == null)
                DebugNull(transform, MonsterComponentData.MONSTER);

            this.monsterSkill = monsterSkill;
        }
        public override NodeState Evaluate()
        {
            if (monster.ExecuteAttack(monsterSkill))
                return NodeState.SUCCESS;
```

```
                Debug.LogFormat("{0} Is Trying To Attack With {1} That Does Not have : {0}, {1}", transform.name,
monsterSkill.ToString());
                return NodeState.FAILURE;
            }
        }
}
```

- ActionRunToPlayer

몬스터가 플레이어를 향해 달려가는, agent 의 속도를 변경해주는 Node

```
using UnityEngine;
using UnityEngine.AI;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionRunToPlayer : MonsterNode
    {
        private RunToPlayerData skill;
        private NavMeshAgent agent;
        private float originalSpeed;
        private float accumTime;

        public ActionRunToPlayer()
        {
            skill = tree.GetMonsterData<RunToPlayerData>(MonsterData.RunToPlayer);
            if (agent == null)
                agent = tree.GetComponentData<NavMeshAgent>(MonsterComponentData.AGENT);
            if (agent == null)
                DebugNull(transform, MonsterComponentData.AGENT);

            originalSpeed = tree.GetMonsterData<MonsterStatData>(MonsterData.MonsterStat).speed;
            accumTime = 0.0f;
        }
        protected override void OnStart()
        {
            agent.speed = skill.speed;
            isStarted = true;
        }
        public override NodeState Evaluate()
        {
            if (!isStarted)
                OnStart();

            Vector3 playerPos = tree.GetBTData<Vector3>(BTData.v3PlayerPosition);
            agent.destination = playerPos;
            float distance = Vector3.SqrMagnitude(playerPos - transform.position);

            if(accumTime<skill.duration)
```

```
            {
                if (distance < skill.stopDistance)
                {
                    OnExit();
                    return NodeState.SUCCESS;
                }
                accumTime += Time.deltaTime;
                return NodeState.RUNNING;
            }

            OnExit();
            return NodeState.SUCCESS;
        }

        protected override void OnExit()
        {
            agent.speed = originalSpeed;
            accumTime = 0.0f;
            isStarted = false;
        }
    }
}
```

- ActionSkillUseable

몬스터의 스킬 사용 가능 여부를 플레이어와의 거리, 쿨타임을 기준으로 확인하는 Node

```
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class ActionSkillUseable : MonsterNode
    {
        float attackableDistance = 0.0f;
        float interval = 0.0f;
        private float lastTime;

        public ActionSkillUseable(float interval, float attackableDistance)
        {
            this.interval = interval;
            this.attackableDistance = attackableDistance;

            lastTime = 0.0f;
        }
        public override NodeState Evaluate()
        {
            float playerDistanceSqr = tree.GetBTData<float>(BTData.fPlayerDistanceSqr);
```

```
            if (playerDistanceSqr > attackableDistance)
                return NodeState.FAILURE;
            if (Time.time - lastTime < interval)
                return NodeState.FAILURE;

            lastTime = Time.time;

            return NodeState.SUCCESS;
        }
    }
}
```

4-5. Composite Node

- Select

하위 노드들 중 하나가 성공할 때 까지 순차적으로 평가하여 하나가 성공하면 즉시 성공으로 간주되는 노드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class Select : Node
    {
        private int current;
        public Select() : base() { }
        public Select(List<Node> childrens) : base(childrens) { }

        protected override void OnStart()
        {
            current = 0;
            isStarted = true;
        }
        public override NodeState Evaluate()
        {
            if (!isStarted)
                OnStart();
            for(int i=current;i<children.Count;i++)
            {
                current = i;
                var child = children[current];

                switch (child.Evaluate())
                {
                    case NodeState.FAILURE:
                        continue;
                    case NodeState.SUCCESS:
```

```
                    return NodeState.SUCCESS;
                case NodeState.RUNNING:
                    return NodeState.RUNNING;
                default:
                    continue;
            }
        }

        return NodeState.FAILURE;
    }
}
```

- Sequence

하위 노드들을 순차적으로 시행하며 모든 하위 노드가 성공하면 전체가 성공으로 간주되는 노드

```
using System.Collections.Generic;

namespace Scripts.BehaviourTrees.Monster
{
    public class Sequence : MonsterNode
    {
        public Sequence() : base() { }
        public Sequence(List<Node> childrens) : base(childrens) { }
        public override NodeState Evaluate()
        {
            bool anyChildIsRunning = false;

            foreach(Node child in children)
            {
                switch(child.Evaluate())
                {
                    case NodeState.FAILURE:
                        return NodeState.FAILURE;
                    case NodeState.SUCCESS:
                        return NodeState.SUCCESS;
                    case NodeState.RUNNING:
                        anyChildIsRunning = true;
                        continue;
                    default:
                        return NodeState.SUCCESS;
                }
            }

            return anyChildIsRunning ? NodeState.RUNNING : NodeState.SUCCESS;
        }
    }
```

```
}
```

4-6. Decorator

- Inverter

하위 노드의 성공과 실패를 반대로 반환하는 노드

```
namespace Scripts.BehaviourTrees.Monster
{
    public class Inverter : Node
    {
        public Inverter(Node node) : base(node){}
        public override NodeState Evaluate()
        {
            switch (children[0].Evaluate())
            {
                case NodeState.RUNNING:
                    return NodeState.RUNNING;
                case NodeState.SUCCESS:
                    return NodeState.FAILURE;
                case NodeState.FAILURE:
                    return NodeState.SUCCESS;
            }

            return NodeState.FAILURE;
        }

    }
}
```

- Repeater

하위 노드를 계속해서 반복하는 노드

```
using System.Collections.Generic;
using UnityEngine;

namespace Scripts.BehaviourTrees.Monster
{
    public class Repeater : Node
    {
        bool repeatOnSuccess;
        bool repeatOnFailure;
        public Repeater(Node child, bool repeatOnSuccess = true, bool repeatOnFailure = true) : base(child)
        {
            this.repeatOnFailure = repeatOnFailure;
            this.repeatOnSuccess = repeatOnSuccess;
        }
```

```
        public Repeater(List<Node> childrens, bool repeatOnSuccess = true, bool repeatOnFailure = true) :
base(childrens)
        {
            this.repeatOnFailure = repeatOnFailure;
            this.repeatOnSuccess = repeatOnSuccess;
        }
        public override NodeState Evaluate()
        {
            if (children.Count != 1)
            {
                Debug.Log(transform.name + "Repeater Node Has No or More Than 1 Child");
                return NodeState.FAILURE;
            }
            switch (children[0].Evaluate())
            {
                case NodeState.RUNNING:
                    return NodeState.RUNNING;
                case NodeState.SUCCESS:
                    if (repeatOnSuccess) return NodeState.RUNNING;
                    return NodeState.SUCCESS;
                case NodeState.FAILURE:
                    if (repeatOnFailure) return NodeState.FAILURE;
                    return NodeState.FAILURE;
            }

            return NodeState.RUNNING;
        }

    }
}
```

WinAPI 개발문서

Monster & Behaviour Tree

목차

1. 기획 내용

땅따먹기와 포켓몬과의 배틀을 컨셉으로 포켓몬의 공격을 회피 및 방어하면서 보유한 포켓몬의 스킬을 활용하여 80% 이상의 땅을 모두 점령하면 승리하는 게임

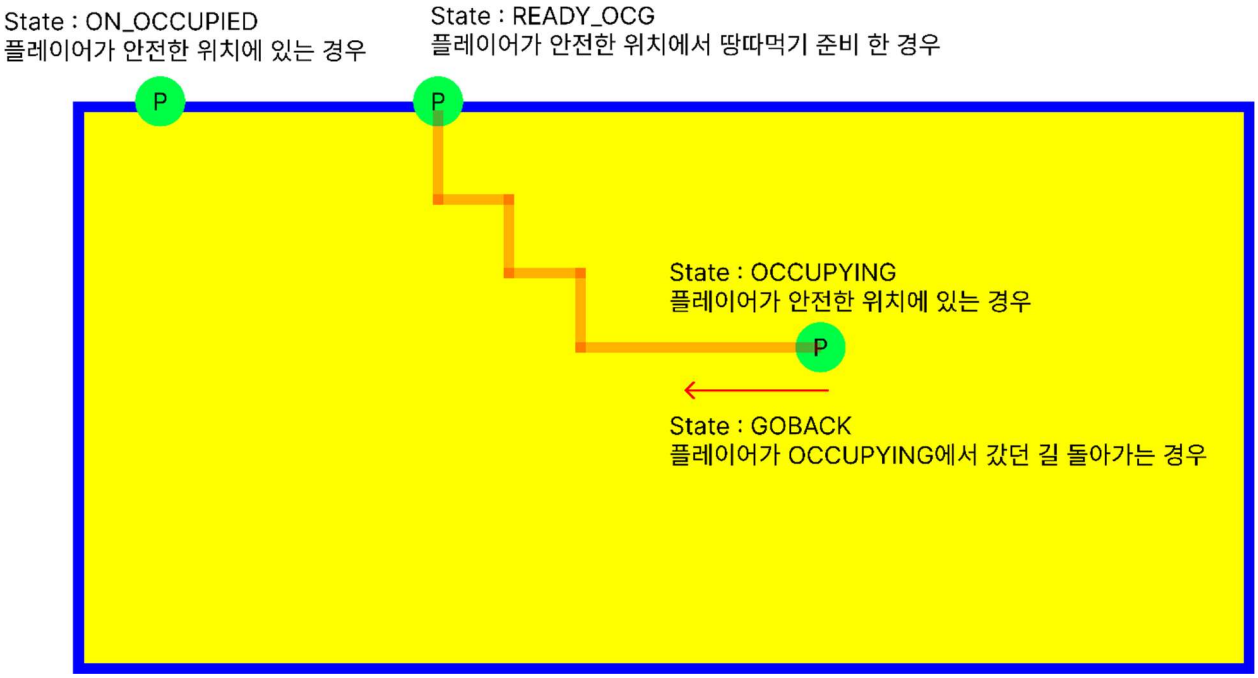
1-1. 플레이어

1-1.1 플레이어 상태

플레이어는 위치와 점령 상태에 따라 4 가지 상태를 가지게 된다.

STATE	설명
ON_OCCUPIED	점령된 땅에 있는 상태
READY_OCG	점령된 땅에서 점령 준비 상태
OCCUPYING	땅을 점령하고 있는 상태
GOBACK	점령 하고 있는 선들을 따라 돌아가는 상태

- ON_OCCPIED
  플레이어가 점령된 땅에서 돌아다닐 수 있으며 몬스터와 충돌되지 않는다.
- READY_OCG
  플레이어가 점령된 땅에서 점령을 시작하기 전으로 점령된 땅 밖으로 나갈 수 있는 상태다.
- OCCUPYING
  플레이어가 점령된 땅을 나와 새롭게 점령하고 있는 상태
- GOBACK
  플레이어가 새롭게 점령하고 있는 상태에서 더 이상 점령을 하지 않고 점령한 역 방향으로 자동으로 돌아가는 상태



1-1.2 플레이어 달리기 및 체력

Shift 를 눌러 달리기를 할 수 있으며 체력이 소모된다. 체력은 플레이어 머리 위에 위치하며 플레이어 상태에 따라 소모 속도와 충전 속도가 상이하다.

STATE	설명	체력 소모 속도	충전 조건	이동속도
ON_OCCUPIED	점령된 땅에 있는 상태		걷기, 멈춤 상태	보통
READY_OCG	점령된 땅에서 점령 준비 상태			
OCCUPYING	땅을 점령하고 있는 상태	ON_OCCUPIED의 2배	멈춤 상태	
GOBACK	점령 하고 있는 선들을 따라 돌아가는 상태			빠름

플레이어 이동은 점령 중 상태가 아닐 시 점령된 선들을 따라 이동할 수 있다. 새롭게 점령할 때는 점령된 땅을 벗어나 파란색 라인으로 점령중인 선이 표시된다.

* 점령중인 땅 이동

* 새롭게 점령중인 땅 이동

1-2. 소유 몬스터 (피카츄)

1-2.1 썬더볼트

Q 를 Hold 하여 플레이어 중심으로 포켓몬을 조종하며, Hold 시간과 비례하여 투사체 개수가 최대 10 개 까지 충전된다. Q 를 Away 시 피카츄 중심으로 랜덤한 방향으로 투사체가 발사된다. 스킬 대기시간은 피카츄 머리 위에 표시된다.

몬스터의 투사체와 충돌 시 두 투사체는 사라진다. 몬스터와 충돌 시 몬스터 이동 속도가 감소되고, 제한된 시간 안에 5 번을 맞추면 몬스터가 마비로 전환된다.

* 썬더볼트 장전

* 썬더볼트 발사

1-2.2 100 만 볼트

썬더볼트와 동일한 기능의 투사체가 100 개 생성된다. R 을 눌러 즉시 사용 가능하며 재사용 시간이 존재하지 않지만 사용 개수가 존재한다.

* 100 만 볼트 발사

1-2. 몬스터

몬스터는 점령되지 않은 땅에서 벽들과 충돌하면서 돌아다니며 상태는 총 3 가지 상태가 존재한다

STATE	설명
NORMAL	기본 상태
PARALYZED	플레이어 투사체와 충돌하여 이동 불가 상태
FURY	마비 상태 해제 이후, 랜덤 시간 후 발생
	움직임과 투사체 속도가 빨라짐

1-2.2 몬스터 공격

- 투사체 공격

몬스터가 돌아다니며 투사체를 놓고가며 천천히 한 방향으로 가다 바로 빠르게 발사된다. 플레이어가 점령상태 때 점령 중인 점들 사이 파란색 선과 충돌시 플레이어의 목숨이 깎이며 플레어는 점령 시작 점으로 복귀된다.

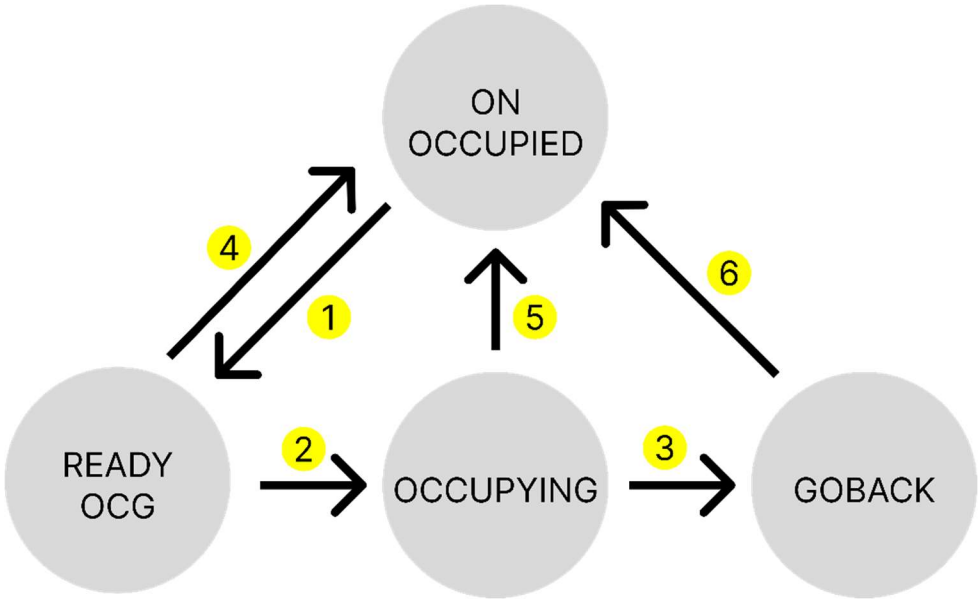
- 늪지대 공격

몬스터는 플레이어 위치에 늪지대를 생성하여 플레이어 이동속도를 저하 시킨다.

* 몬스터 투사체, 늪지대 공격

2. Player FSM

2-1. 다이어그램



1. 플레이어가 점령된 땅에서 점령 키를 PRESS 하여 준비
2. 점령 준비 상태에서 점령된 땅을 나와 새롭게 점령하는 상태(점령 키 HOLD)

3. 새롭게 점령 중인 상태에서 점령 키를 AWAY 하여 새롭게 점령한 점 순서대로 빠르게 자동으로 복귀
4. 점령 준비 상태에서 점령 키 AWAY
5. 점령한 땅을 다시 만나 새로운 땅 점령 완료
6. 새롭게 점령한 모든 점을 복귀하여 시작점으로 돌아온 상태

2-2. 코드

2-2.1 Player FSM

```
void CPlayer::PlayerStateUpdate()
{
    switch (eState)
    {
    case PLAYER_STATE::ON_OCCUPIED:
            StateOnOccupied();
            break;
    case PLAYER_STATE::READY_OCG:
            StateReadyOccupying();
            break;
    case PLAYER_STATE::OCCUPYING:
            StateOccupying();
            break;
    case PLAYER_STATE::GOBACK:
            StateGoBack();
            break;
    }
}
```

2-2.2 ON_OCCUPIED State

점령된 땅에 있는 상태로 달리기 상관 없이 지속적으로 체력이 찬다.

```
void CPlayer::StateOnOccupied()
{
    if (fStamina < 50)
    {
            fStamina += fRecoverStamina * fDT;
            if (fStamina > 50)
                    fStamina = 50;
    }
}
```

2-2.3READY_OCG State

점령 준비 상태로 점령된 땅을 돌아다니며 동일 State 를 유지할 수 있고 점령되지 않은 땅 방향으로 이동하여 OCCUPYING State 으로 전환될 수 있다.

```
void CPlayer::StateReadyOccupying()
{
    // 플레이어가 점령 준비 상태에서 새롭게 점령을 출발했는지 아니면 아직 점령된 땅을 이동하는지 확인
    OCDPoint* ocdPoint = GetOCDVec2();
    OCDPoint* temp = ocdPoint;
    do
    {
            if (ocdPoint->pPoint.AroundEqual(pCenter))
            {
                    pOCGPoint[0] = ocdPoint->pPoint;
                    OCGonOCD = true;
                    break;
            }
            else
                    OCGonOCD = false;
            ocdPoint = ocdPoint->nxt;
    } while (ocdPoint != temp);

    if (!OCGonOCD)
            pOCGPoint[0] = vCenter;

    ocdPoint = ocdPoint->nxt;
    bCollect = true;
    eDirStart = DIRECTION::END;
}
```

2-2.4 OCCUPYING State

플레이어가 점령된 땅을 벗어나 새롭게 점령하며 방향을 변경할 때 마다 점들을 링크드 리스트 형태로 수집한다 : CollectOCGPoints()

시작 위치가 아닌 다른 위치의 점령된 땅을 다시 만나면 점령이 완료되어 링크드 리스트형태의 점령된 땅 사이에 새롭게 점령된 링크드 리스트가 추가된다.

```
void CPlayer::StateOccupying()
{
    // 새롭게 점령하고 있는 땅들 포인트 수집 하다가 플레이어가 점령된 땅에 다시 닿을 시 수집된 점들 새롭게 추가
    if (eDirStart == DIRECTION::END)
    {
            eDirStart = eDirCrnt;
            ocgCnt = 1;
    }
```

```
    CollectOCGPoints();

    if (IsPlayerOnOCD())
    {

            eState = PLAYER_STATE::ON_OCCUPIED;

            OCDPoint* ocdCrnt = GetOCDVec2();
            OCDPoint* temp = ocdCrnt;
            do
            {
                    if (IsBetweenX(ocdCrnt->pPoint, ocdCrnt->nxt->pPoint, pOCGPoint[ocgCnt]))
                    {
                            if (abs(ocdCrnt->pPoint.y - pOCGPoint[ocgCnt].y) <= 1
                                    && abs(ocdCrnt->pPoint.y - pOCGPoint[ocgCnt].y) >= 0)
                                    pOCGPoint[ocgCnt].y = ocdCrnt->pPoint.y;
                    }
                    if (IsBetweenY(ocdCrnt->pPoint, ocdCrnt->nxt->pPoint, pCenter))
                    {
                            if (abs(ocdCrnt->pPoint.x - pOCGPoint[ocgCnt].x) <= 1
                                    && abs(ocdCrnt->pPoint.x - pOCGPoint[ocgCnt].x) >= 0)
                                    pOCGPoint[ocgCnt].x = ocdCrnt->pPoint.x;
                    }
            } while (ocdCrnt != temp);
            ocgCnt++;

            CScene* crntScene = CSceneMgr::GetInst()->GetCurrentScene();
            crntScene->GetOCD()->FinishOCG(GetOCGPack());
            OCGonOCD = false;
            ocgCnt = 0;
    }
}
```

2-2.5 GOBACK State

플레이어가 해당 State 에서 점령된 땅을 만나면 ON_OCCUPIED State 로 전환된다.

위 사항이 아니면 마지막으로 새롭게 점령된 점 방향으로 이동하며 만나면 해당 점을 삭제 후 다시 마지막 점으로 이동을 반복한다.

```
void CPlayer::StateGoBack()
{
            bRetreat = true;
            if (IsPlayerOnOCD())
            {
                    eState = PLAYER_STATE::ON_OCCUPIED;
                    ocgCnt = 0;
```

```
		}
		else		MoveGoBack();
}

void CPlayer::MoveGoBack()
{
		Point vLastOcg = pOCGPoint[ocgCnt - 1];
		Point v_iPos = { Vec2(vCenter.x, vCenter.y).ToPoint()};
		if (eState == PLAYER_STATE::OCCUPYING)
		{
				vCenter = pOCGPoint[ocgCnt].ToVec2();
				eState = PLAYER_STATE::GOBACK;
		}
		else if (eState==PLAYER_STATE::GOBACK)
		{
				if (ocgCnt == 0)
				{
						vCenter = pOCGPoint[0].ToVec2();
				}
				if ((int)vLastOcg.x == (int)v_iPos.x)
				{
						if ((int)vLastOcg.y == (int)v_iPos.y)
								ocgCnt--;
						else if ((int)vLastOcg.y > (int)v_iPos.y)
								MoveDown();
						else
								MoveUp();
				}
				else if ((int)vLastOcg.y == (int)v_iPos.y)
				{
						if ((int)vLastOcg.x == (int)v_iPos.x)
								ocgCnt--;
						else if ((int)vLastOcg.x > (int)v_iPos.x)
								MoveRight();
						else
								MoveLeft();
				}
		}
		pOCGPoint[ocgCnt] = v_iPos;
		bCollect = false;
}
```

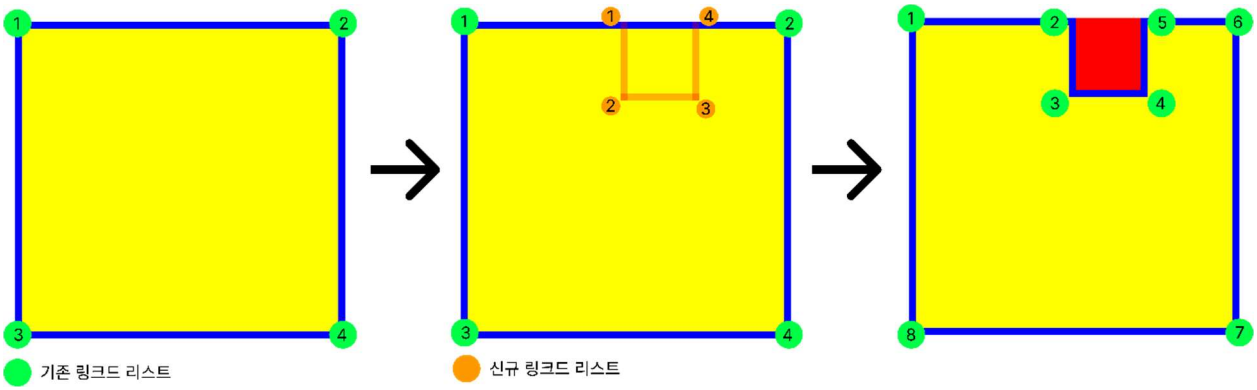
3. 링크드 리스트

3-1. 링크드 리스트로 점령 관리

점령된 땅은 링크드 리스트로 각 점들을 순차적으로 연결되어 있다. 새롭게 점령 한 땅은 기존 점령된 땅의 링크드 리스트 중간에 삽입된다. 새로운 링크드 리스트가 삽입 될 때 기존 링크드 리스트 점이 삭제될 수 있다.

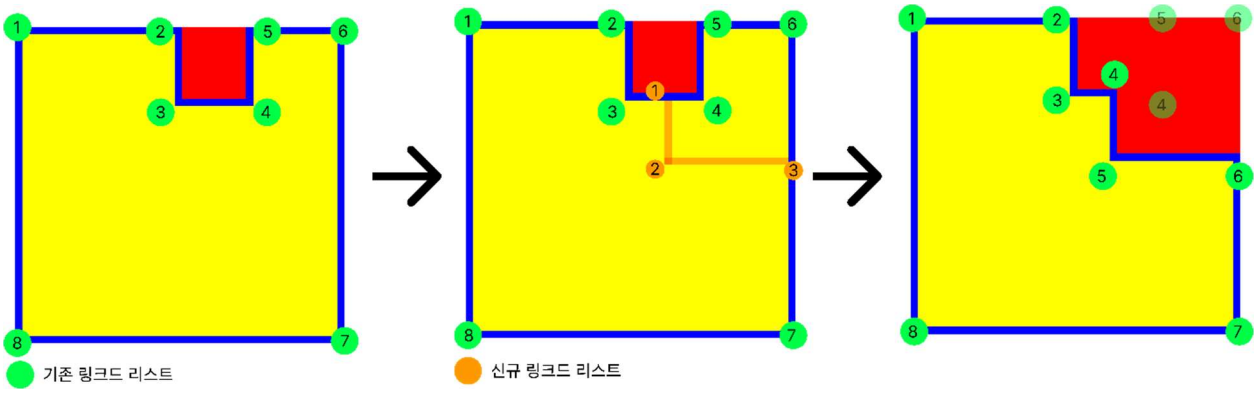
3-1.1 기존 링크드 리스트에 신규 링크드 리스트 삽입

기존 링크스 리스트 점 1 과 2 사이에 신규 링크드 리스트 1~4 가 삽입되어 최종 점 8 개의 링크드 리스트 생성



3-1.2 기존 링크드 리스트 일부 점 삭제 및 신규 링크드 리스트 삽입

기존 링크드 리스트 점 3 과 7 사이에 신규 링크드 리스트 1~3 가 삽입 되면서 기존 링크드 리스트 4, 5, 6 이 삭제 되고 점 8 개의 링크드 리스트 생성



3-2. 코드

3-2.1 점 구조체

점 구조체는 자신의 점 앞, 뒤의 점의 주소값을 변수로 가지고 있다. 자기

자신 앞뒤로 점을 삽입할 수 있는 함수를 가지고 있어 삽입되어야 하는 위치에서 새로운 링크드 리스트 시작점을 쉽게 추가 할 수 있다.

- 헤더파일

```
struct OCDPoint
{
		Point pPoint;
		OCDPoint* nxt;
		OCDPoint* prv;

public:
		OCDPoint();
		OCDPoint(Vec2 _vec2);
		void InsertNxt(OCDPoint* _OCDPoint);
		void InsertPrv(OCDPoint* _OCDPoint);
		void InsertNxt(Vec2 _vec2);
		void InsertPrv(Vec2 _vec2);
		void DeleteSelf();
};
```

- 정의 파일

```
OCDPoint::OCDPoint()
		:pPoint(0, 0)
		, nxt(nullptr)
		, prv(nullptr)
{}
OCDPoint::OCDPoint(Vec2 _vec2)
{
		pPoint = _vec2;
		nxt = nullptr;
		prv = nullptr;
}
void OCDPoint::InsertNxt(OCDPoint* _OCDPoint)
{
		_OCDPoint->prv = this;
		_OCDPoint->nxt = nxt;
		nxt->prv = _OCDPoint;
		nxt = _OCDPoint;
}
void OCDPoint::InsertPrv(OCDPoint* _OCDPoint)
{
		_OCDPoint->nxt = this;
		_OCDPoint->prv = prv;
		prv->nxt = _OCDPoint;
		prv = _OCDPoint;
}
void OCDPoint::InsertNxt(Vec2 _vec2)
{
		OCDPoint* _OCDPoint = new OCDPoint(_vec2);
```

```
		InsertNxt(_OCDPoint);
}
void OCDPoint::InsertPrv(Vec2 _vec2)
{
		OCDPoint* _OCDPoint = new OCDPoint(_vec2);
		InsertPrv(_OCDPoint);
}
void OCDPoint::DeleteSelf()
{
		prv->nxt = nxt;
		nxt->prv = prv;
		delete this;
}
```

3-2.2 초기 4개 점 설정

```
void COcuppied::SetStartOCD()
{
		RECT rect = CCore::GetInst()->GetOCDTotalSize();

		OCDPoint* temp = new OCDPoint(Vec2(rect.left, rect.top));

		fTotalArea = (rect.right - rect.left) * (rect.bottom - rect.top);

		// >> : set head
		ocdVec2 = temp;
		ocdVec2->nxt = ocdVec2;
		ocdVec2->prv = ocdVec2;
		// << : set head

		// >> : insert start points
		ocdVec2->InsertNxt(Vec2(rect.right, rect.top));
		ocdVec2 = ocdVec2->nxt;
		ocdVec2->InsertNxt(Vec2(rect.right, rect.bottom));
		ocdVec2 = ocdVec2->nxt;
		ocdVec2->InsertNxt(Vec2(rect.left, rect.bottom));
		// << : insert start points
}
```

3-2.3 신규 링크드 리스트 점 수집

```
void CPlayer::CollectOCGPoints( )
{
		if (eDirBefore != eDirCrnt && eDirBefore != OppositeDirection(eDirCrnt))
		{
				pOCGPoint[ocgCnt] = vCenter;
				if (abs(pOCGPoint[ocgCnt].x - pOCGPoint[ocgCnt - 1].x) == 1)
				{
						pOCGPoint[ocgCnt].x = pOCGPoint[ocgCnt - 1].x;
```

```
                }
                else if (abs(pOCGPoint[ocgCnt].y - pOCGPoint[ocgCnt - 1].y) == 1)
                {
                        pOCGPoint[ocgCnt].y = pOCGPoint[ocgCnt - 1].y;
                }
                ocgCnt++;
        }
        else
        {
                pOCGPoint[ocgCnt] = vCenter.ToPoint();
        }
}
```

3-2.4 신규 링크드 리스트 추가 위치 확인 및 삭제/삽입

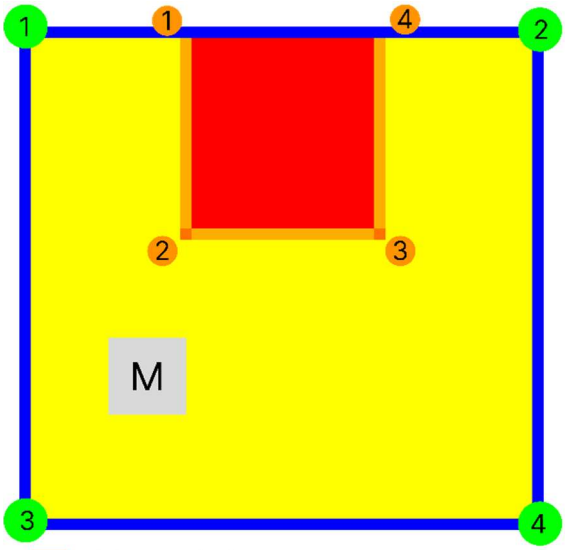
신규 링크드 리스트 완료 시 실행되는 함수로 기존 점 삭제, 신규 점 추가, 수정된 링크드 리스트 업데이트

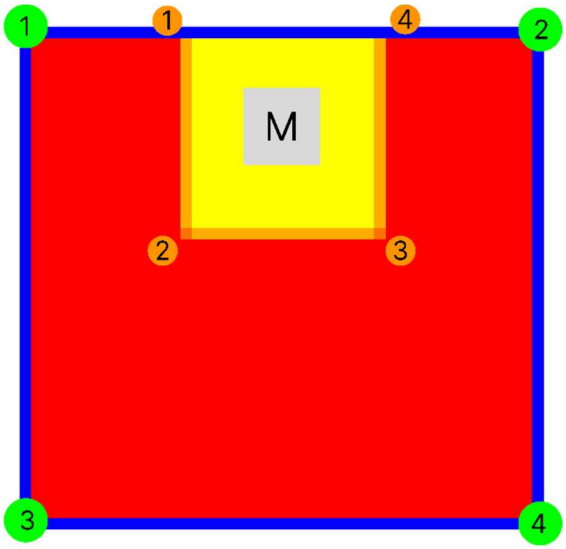
```
void COcuppied::FinishOCG(PackOCG _ocgPack)
{
        ocgPack = _ocgPack;
        CScene* crntScene = CSceneMgr::GetInst()->GetCurrentScene();
        vEnemyCen = crntScene->GetEnemy()->GetPosition();

        if(!ocgPack.bSameDir)
        {
                if (!IsEnemyInOCG())
                        ocgPack.bClockwise = !ocgPack.bClockwise;
                ConfigDeletePoint();
        }
        else
        {
                OCGSameDirection();
        }
        if (ocdDeleteStart == nullptr || ocdDeleteEnd == nullptr || ocdAddStart == nullptr) return;

        DeleteOCDPoint();
        AddOCGPoint();

        bStartOCGonOCD = false;
        bEndOCGonOCD = false;

        StraightOCD();
        UpdateOCDArea();
        if (fTotalPercent> FinishRatio)
        {
                bFinishGame = true;
                fDTFinishGame = 0;
        }
```

```
}
```

3-2.5 몬스터가 신규 링크드 리스트 내 외부 확인

몬스터의 위치에 따라 점령되는 땅이 상이하다. 새롭게 점령된 땅 밖에 몬스터가 있는 경우 새롭게 점령된 땅 내부가 점령되고 밖에 있는 경우 새롭게 점령된 땅 외의 땅들이 점령된다.






```
bool COcuppied::IsEnemyInOCG()
{
		int enemyLineCnt = 0;

		//몬스터 좌우위아래 중 ocg 선이 홀수면 in 짝수면 out
		DIRECTION startCnt = DIRECTION::END;
		for (int i = 0; i < ocgPack.ocgCnt; i++)
		{
			if ((startCnt == DIRECTION::END || startCnt == DIRECTION::RIGHT) && vEnemyCen.x < 
ocgPack.ocgPoint[i].x)
			{
				if (IsBetweenY(ocgPack.ocgPoint[i], ocgPack.ocgPoint[(i + 1)%ocgPack.ocgCnt], 
vEnemyCen))
				{
					enemyLineCnt++;
					startCnt = DIRECTION::RIGHT;
				}
			}
			if ((startCnt == DIRECTION::END || startCnt == DIRECTION::LEFT) && vEnemyCen.x > 
ocgPack.ocgPoint[i].x)
			{
				if (IsBetweenY(ocgPack.ocgPoint[i], ocgPack.ocgPoint[(i + 1) % ocgPack.ocgCnt], 
vEnemyCen))
```

```
                                        {
                                                enemyLineCnt++;
                                                startCnt = DIRECTION::LEFT;
                                        }
                                }
                                if ((startCnt == DIRECTION::END || startCnt == DIRECTION::DOWN) && vEnemyCen.y <
ocgPack.ocgPoint[i].y)
                                {
                                        if (IsBetweenX(ocgPack.ocgPoint[i], ocgPack.ocgPoint[(i + 1) % ocgPack.ocgCnt],
vEnemyCen))
                                        {
                                                enemyLineCnt++;
                                                startCnt = DIRECTION::DOWN;
                                        }
                                }
                                if ((startCnt == DIRECTION::END|| startCnt == DIRECTION::UP) && vEnemyCen.y >
ocgPack.ocgPoint[i].y)
                                {
                                        if (IsBetweenX(ocgPack.ocgPoint[i], ocgPack.ocgPoint[(i + 1) % ocgPack.ocgCnt],
vEnemyCen))
                                        {
                                                enemyLineCnt++;
                                                startCnt = DIRECTION::UP;
                                        }
                                }
                        }
                        if (enemyLineCnt % 2 != 0)
                        {
                                return true;
                        }
                        return false;
}
```

3-2.6 기존 링크드 리스트 삭제 점 확인

- 신규 링크드 리스트의 시작점과 끝점 사이에 있는 기존 점의 시작과 끝 확인

```
void COcuppied::ConfigDeletePoint()
{
        ocdDeleteStart = nullptr;
        ocdDeleteEnd = nullptr;
        ocdAddStart = nullptr;
        OCDPoint* temp = ocdVec2;
        if (ocgPack.bClockwise)
        {
                do
                {
                        if (IsOnXorYAxis(ocdVec2->pPoint, ocdVec2->nxt->pPoint, ocgPack.ocgPoint[0])
```

```
                                        &&ocdDeleteStart==nullptr)
                                {
                                        ocdDeleteStart = ocdVec2->nxt;
                                        ocdAddStart = ocdVec2;
                                }
                                if (IsOnXorYAxis(ocdVec2->pPoint, ocdVec2->nxt->pPoint,
ocgPack.ocgPoint[ocgPack.ocgCnt - 1]))
                                {
                                        ocdDeleteEnd = ocdVec2;
                                }
                                if (ocdVec2->pPoint.AroundEqual(ocgPack.ocgPoint[0]))
                                {
                                        ocdDeleteStart = ocdVec2;
                                        ocdAddStart = ocdVec2->prv;
                                }
                                ocdVec2 = ocdVec2->nxt;
                        } while (temp != ocdVec2);
                }
                else //ocgPack.bClockwise == true
                {
                        do
                        {
                                if (IsOnXorYAxis(ocdVec2->pPoint, ocdVec2->nxt->pPoint, ocgPack.ocgPoint[0])
                                        &&ocdDeleteStart==nullptr)
                                {
                                        ocdDeleteStart = ocdVec2;
                                        ocdAddStart = ocdVec2->nxt;
                                }
                                if (IsOnXorYAxis(ocdVec2->pPoint, ocdVec2->nxt->pPoint,
ocgPack.ocgPoint[ocgPack.ocgCnt - 1]))
                                {
                                        ocdDeleteEnd = ocdVec2->nxt;
                                }
                                if (ocdVec2->pPoint.AroundEqual(ocgPack.ocgPoint[0]))
                                {
                                        ocdDeleteStart = ocdVec2->prv;
                                        ocdAddStart = ocdVec2;
                                }
                                ocdVec2 = ocdVec2->nxt;
                        } while (temp != ocdVec2);
                }
}
```

- 기존 링크드 리스트 점 삭제

ConfigDeletePoint()에서 검토된 삭제되어야 하는 점 시작과 끝 순차적으로 삭제

```
void COcuppied::DeleteOCDPoint()
```

```
{
	OCDPoint* temp = ocdDeleteStart;
	//시작점과 끝점 사이에 아무 점이 없고 몬스터가 안에 없을 때
	if (ocdDeleteStart == ocdDeleteEnd&&!ocdDeleteStart->pPoint.AroundEqual(ocdDeleteEnd->pPoint))
	{
		ocdDeleteStart->DeleteSelf();
		ocdDeleteStart = nullptr;
	}
	//시작점과 끝점 사이에 아무 점이 없고 몬스터가 안에 있을 때
	else if (IsEnemyInOCG()&&(ocdDeleteStart == ocdAddStart || ocdDeleteEnd == ocdAddStart))
	{
		if(ocgPack.bClockwise)
		{
			while (temp != ocdDeleteEnd)
			{
				temp = ocdDeleteStart->nxt;
				ocdDeleteStart->DeleteSelf();
				ocdDeleteStart = temp;
			}
			ocdDeleteEnd->DeleteSelf();
		}
		else //ocgPack.bClockwise==false;
		{
			while (temp != ocdDeleteEnd)
			{
				temp = ocdDeleteStart->prv;
				ocdDeleteStart->DeleteSelf();
				ocdDeleteStart = temp;
			}
			ocdDeleteEnd->DeleteSelf();
		}

		bDeleteAllPoints = true;
	}
	else if (ocgPack.bClockwise)
	{
		OCDPoint* ocdEndNxt = ocdDeleteEnd->nxt;
		while(temp!=ocdEndNxt)
		{
			temp = temp -> nxt;
			ocdDeleteStart->DeleteSelf();
			ocdDeleteStart = temp;
		}
	}
	else //ocgPack.bClockwise==false
	{
		OCDPoint* ocdEndPrv = ocdDeleteEnd->prv;
		while (temp != ocdEndPrv)
		{
```

```
                        temp = temp->prv;
                        ocdDeleteStart->DeleteSelf();
                        ocdDeleteStart = temp;
                }
        }
}
```

- 신규 링크드 리스트 점 추가

기존 링크드 리스트 사이에 신규 링크드 리스트 점 추가

```
void COcuppied::AddOCGPoint()
{
        int i = 0;

        if (bStartOCGonOCD) i = 1;
        if (bEndOCGonOCD) ocgPack.ocgCnt--;
        if (!bDeleteAllPoints)
        {
                if (ocgPack.bClockwise)
                {
                        for (; i < ocgPack.ocgCnt; i++)
                        {
                                ocdAddStart->InsertNxt(ocgPack.ocgPoint[i].ToVec2());
                                ocdAddStart = ocdAddStart->nxt;
                        }
                }
                else //ocg.PackbClockwise==false
                {
                        for (; i < ocgPack.ocgCnt; i++)
                        {
                                ocdAddStart->InsertPrv(ocgPack.ocgPoint[i].ToVec2());
                                ocdAddStart = ocdAddStart->prv;
                        }
                }
                ocdVec2 = ocdAddStart;
        }
        else
        {
                ocdVec2= new OCDPoint(ocgPack.ocgPoint[2]);
                ocdVec2->prv = new OCDPoint(ocgPack.ocgPoint[1]);
                ocdVec2->nxt = ocdVec2->prv;
                ocdVec2->prv->prv = ocdVec2;
                ocdVec2->prv->nxt = ocdVec2;

                ocdVec2->InsertNxt(ocgPack.ocgPoint[0]);

                for (int i = 3; i < ocgPack.ocgCnt; i++)
```

```
                {
                        ocdVec2->InsertNxt(ocgPack.ocgPoint[i].ToVec2());
                        ocdVec2 = ocdVec2->nxt;
                }

                bDeleteAllPoints = false;
        }
}
```

- 점령된 땅 업데이트

새로운 링크드 리스트를 기준으로 점령된 땅 정보 업데이트 (Render, 점령률 확인)

```
void COcuppied::UpdateOCDArea()
{
        OCDPoint* temp = ocdVec2;
        float fSum1 = 0;
        float fSum2 = 0;
        do
        {
                fSum1 += ocdVec2->pPoint.x * ocdVec2->nxt->pPoint.y;
                fSum2 += ocdVec2->pPoint.y * ocdVec2->nxt->pPoint.x;
                ocdVec2 = ocdVec2->nxt;
        } while (ocdVec2 != temp);
        fOCDArea = (fSum1 - fSum2) / 2;

        fTotalPercent = (fTotalArea - fOCDArea) / fTotalArea;
}
```